

Proceedings of the Workshops of  
the EDBT/ICDT 2014 Joint Conference

K. Selçuk Candan  
Arizona State University, USA

Sihem Amer-Yahia  
CNRS – LIG, France

Nicole Schweikardt  
University of Frankfurt, Germany

Vassilis Christophides  
University of Crete & FORTH-ICS, Greece

Vincent Leroy  
University of Grenoble – CNRS, France

March 28th, 2014

# Contents

Message from the Chairs . . . . .	iii
Algorithms for MapReduce and Beyond (BeyondMR) . . . . .	1
Scheduling MapReduce Jobs on Unrelated Processors . . . . .	2
Binary Theta-Joins using MapReduce: Efficiency Analysis and Improvements . . . . .	6
On the design space of MapReduce ROLLUP aggregates . . . . .	10
Determining the k in k-means with MapReduce . . . . .	19
Tagged Dataflow: a Formal Model for Iterative Map-Reduce . . . . .	29
Processing Regular Path Queries on Giraph . . . . .	37
Graph-Parallel Entity Resolution using LSH & IMM . . . . .	41
Modular Data Clustering - Algorithm Design beyond MapReduce . . . . .	50
Bidirectional Transformations (BX) . . . . .	60
Preface to the Third International Workshop on Bidirectional Transformations . . . . .	61
Implementing a Bidirectional Model Transformation Language as an Internal DSL in Scala . . . . .	63
Towards a Framework for Multidirectional Model Transformations . . . . .	71
Formalizing Semantic Bidirectionalization with Dependent Types . . . . .	75
BenchmarX . . . . .	82
Towards a Repository of Bx Examples . . . . .	87
Intersection Schemas as a Dataspace Integration Technique . . . . .	92
Bidirectional Transformations in Database Evolution: A Case Study “At Scale” . . . . .	100
Entangled State Monads . . . . .	108
Spans of lenses . . . . .	112
Energy Data Management (EnDM) . . . . .	119
Pipeline Production Data Model . . . . .	120
Renewable Energy Data Sources in the Semantic Web with OpenWatt . . . . .	128
A Generic Ontology for Prosumer-Oriented Smart Grid . . . . .	134
Computing Electricity Consumption Profiles from Household Smart Meter Data . . . . .	140
ECAST: A Benchmark Framework for Renewable Energy Forecasting Systems . . . . .	148
Energy Data Management: Where Are We Headed? (panel) . . . . .	156
Exploratory Search in Databases and the Web (ExploreDB) . . . . .	157
Exploratory Search in Databases and the Web . . . . .	158
Exploring Big Data using Visual Analytics . . . . .	160
On the Suitability of Skyline Queries for Data Exploration . . . . .	161
Hippalus: Preference-enriched Faceted Exploration . . . . .	167
The DisC Diversity Model . . . . .	173
Exploring RDF/S Evolution using Provenance Queries . . . . .	176
Skyline Ranking à la IR . . . . .	182
Multi-Engine Search and Language Translation . . . . .	188
Querying Graph Structured Data (GraphQ) . . . . .	191
An Event-Driven Approach for Querying Graph-Structured Data Using Natural Language . . . . .	192
GraphMCS: Discover the Unknown in Large Data Graphs . . . . .	200



Graph-driven Exploration of Relational Databases for Efficient Keyword Search . . . . .	208
Implementing Iterative Algorithms with SPARQL . . . . .	216
A Map-Reduce algorithm for querying linked data based on query decomposition into stars . . . . .	224
Performance optimization for querying social network data . . . . .	232
Frequent Pattern Mining from Dense Graph Streams . . . . .	240
Linked Web Data Management (LWDM) . . . . .	248
Quantifying the Connectivity of a Semantic Warehouse . . . . .	249
Scalable Numerical SPARQL Queries over Relational Databases . . . . .	257
Similarity Recognition in the Web of Data . . . . .	263
Mining of Diverse Social Entities from Linked Data . . . . .	269
TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples . . . . .	275
Multimodal Social Data Management (MSDM) . . . . .	279
Social Data and Multimedia Analytics for News and Events Applications . . . . .	280
Event Identification and Tracking in Social Media Streaming Data . . . . .	282
Recommendation of Multimedia Objects for Social Network Applications . . . . .	288
Estimating Completeness in Streaming Graphs . . . . .	294
Mining Urban Data (MUD) . . . . .	300
Mining Trajectory Data for Discovering Communities of Moving Objects . . . . .	301
Mobile Sensing Data for Urban Mobility Analysis: A Case Study in Preprocessing . . . . .	309
Crowd Density Estimation for Public Transport Vehicles . . . . .	315
Traffic Incident Detection Using Probabilistic Topic Model . . . . .	323
Predictive Trip Planning – Smart Routing in Smart Cities . . . . .	331
Addressing the Sparsity of Location Information on Twitter . . . . .	339
Efficient Dissemination of Emergency Information using a Social Network . . . . .	347
Crowdsourcing turning restrictions for OpenStreetMap . . . . .	355
Big data analytics for smart mobility: a case study . . . . .	363
Smart Applications for Smart City: a Contribution to Innovation . . . . .	365
Analysis of Relationships Between Road Traffic Volumes and Weather: Exploring Spatial Variation . . . . .	367
SiCi Explorer: Situation Monitoring of Cities in Social Media Streaming Data . . . . .	369
A Cascading Wavelet-Feed Forward Neural Network Approach for Forecasting Traffic Flow . . . . .	371
Combining a Gauss-Markov model and Gaussian process for traffic prediction in Dublin city center . . . . .	373
Sensing Urban Soundscapes . . . . .	375
Privacy and Anonymity in the Information Society (PAIS) . . . . .	383
A Hybrid Approach for Privacy-preserving Record Linkage . . . . .	384
Clustering-based Multidimensional Sequence Data Anonymization . . . . .	385
Efficient Multi-User Indexing for Secure Keyword Search . . . . .	390
Community Detection in Anonymized Social Networks . . . . .	396
Secure Multi-Party linear Regression . . . . .	406
Data Anonymization: The Challenge from Theory to Practice . . . . .	415
A Privacy Preserving Model for Ownership Indexing in Distributed Storage Systems . . . . .	416

# Message from the Chairs

We are delighted to present to you, on behalf of the entire conference organizing committee and the workshop organizers, the proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference, held on March 28, 2014, in Athens, Greece.

International Conference on Extending Database Technology (EDBT) and International Conference on Database Theory (ICDT) are two prestigious forums for the exchange of the latest research results in data management and the theoretical foundations of database systems. While having the same overarching goal of presenting cutting-edge results, ideas, techniques, and theoretical advances in databases, the workshops of the EDBT/ICDT joint conference are separately tasked by focusing on emerging topics that complement the areas covered by the main technical program.

This year, our program includes workshops focusing on eight exciting topics:

- *Algorithms for MapReduce and Beyond (BeyondMR)* workshop, aiming to explore algorithms and computational models for systems that need large scale parallelization and systems designed to support efficient parallelization and fault tolerance,
- *Bidirectional Transformations (BX)* workshop, bringing together researchers and practitioners, established and new, interested in bidirectional transformations from different perspectives,
- *Energy Data Management (EnDM)* workshop, focusing on conceptual and system architecture issues related to the management of very large-scale data sets specifically in the context of the energy domain,
- *Exploratory Search in Databases and the Web (ExploreDB)* workshop, aiming to promote novel discovery methods that provide highly expressive discovery capabilities over large amounts of entity-relationship data, which are yet intuitive for end-users,
- *Linked Web Data Management (LWDM)* workshop, aiming at stimulating participants to discuss about data management issues related to the Linked Data and the relationships with other Semantic Web technologies, and at the same time proposes a glance at new issues,
- *Multimodal Social Data Management (MSDM)* workshop, bringing together experts in social network analysis, natural language processing, multimodal data management and integration, scalable data analysis, machine learning, to discuss how research contributions in different computer science areas can help better explain social data and build new applications,
- *Privacy and Anonymity in the Information Society (PAIS)* workshop, which provides a platform for researchers and practitioners from computer science and other fields that are interacting with computer science in the privacy area, such as statistics, healthcare informatics, and law, to discuss and present current research challenges and advances in data privacy and anonymity research, and
- *Querying Graph Structured Data (GraphQ)* workshop, which aims to encourage discussions about how to efficiently and effectively support graph queries in different application domains and seeks to provide the opportunity for cross-fertilization among teams working on graph-structured data, with a particular focus on the querying issues.

Before concluding, we would like to acknowledge those who have contributed to the success of the workshops program. First of all, we would like to thank all workshop organizers who have put together an exciting program as well as to all authors who submitted their works to the workshops. We specially thank the authors of the accepted papers and the invited speakers who presented their works in the workshops program. Needless to say, we are grateful to the members of the workshop program committees and external reviewers who have helped put together a high-quality workshops program and we would like to acknowledge the conference organizers and many student volunteers for their invaluable help at various stages of the process. We would also like to give our thanks to the sponsors who have financially supported the workshops and the editors of the CEUR Workshop Proceedings (CEUR-WS.org) who have agreed to host these proceedings.

Sincerely,

K. Selçuk Candan, *Workshops Chair*

Sihem Amer-Yahia and Nicole Schweikardt, *EDBT and ICDT Program Chairs*

Vassilis Christophides, *General Chair*

# Algorithms for MapReduce and Beyond (BeyondMR)

Foto N. Afrati (National Technical University of Athens, Greece)  
Phokion G. Kolaitis (UC Santa Cruz & IBM Research, USA)  
Jeffrey D. Ullman (Stanford University, USA)

# Scheduling MapReduce Jobs on Unrelated Processors\*

D. Fotakis  
National Technical University  
of Athens  
fotakis@cs.ntua.gr

I. Milis  
Athens University of  
Economics and Business  
milis@aueb.gr

E. Zampetakis  
National Technical University  
of Athens  
mzampet@corelab.ntua.gr

G. Zois  
Université Pierre et Marie  
Curie and Athens University of  
Economics and Business  
Georgios.Zois@lip6.fr

## ABSTRACT

MapReduce framework is established as the standard approach for parallel processing of massive amounts of data. In this work, we extend the model of MapReduce scheduling on unrelated processors (Moseley et al., SPAA 2011) and deal with the practically important case of jobs with any number of Map and Reduce tasks. We present a polynomial-time  $(32 + \epsilon)$ -approximation algorithm for minimizing the total weighted completion time in this setting. To the best of our knowledge, this is the most general setting of MapReduce scheduling for which an approximation guarantee is known. Moreover, this is the first time that a constant approximation ratio is obtained for minimizing the total weighted completion time on unrelated processors under a nontrivial class of precedence constraints.

## Keywords

MapReduce, Scheduling, Unrelated Processors

## 1. INTRODUCTION

Scheduling in MapReduce environments has become increasingly important during the last years, as MapReduce has been established as the standard programming model to implement massive parallelism in large data centers [5]. Applications of MapReduce such as search indexing, web analytics and data mining, involve the concurrent execution of several MapReduce jobs on a system like Google's MapReduce or Apache Hadoop. When a MapReduce job is executed, a number of Map and Reduce tasks are created.

\*This work was supported by the project Handling Uncertainty in Data Intensive Applications, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the program THALES, and by the project Heracleitus II.

Each Map task operates on a portion of the input elements, translating them into a number of key-value pairs. Next, all key-value pairs are transmitted to the Reduce tasks, so that all pairs with the same key are available together at the same task. The Reduce tasks operate on the key-value pairs, combine the values associated with a key, and generate the final result. In addition to the many practical applications of MapReduce, there has been a significant interest in developing appropriate cost models and a computational complexity theory for MapReduce computation (see e.g., [3, 6]), in understanding the basic principles underlying the design of efficient MapReduce algorithms (see e.g., [1, 7]), and in obtaining upper and lower bounds on the performance of MapReduce algorithms for some fundamental computational problems (see e.g. [2] and the references therein).

**Motivation and Previous Work.** Many important advantages of MapReduce are due to the fact that the Map tasks or the Reduce tasks can be executed in parallel and essentially independent from each other. However, to best exploit massive parallelism available in typical MapReduce systems, one has to carefully allocate and schedule Map and Reduce tasks to actual processors (or computational resources, in general). This important and delicate task is performed in a centralized manner, by a process running in the master node. A major concern of the scheduler, among others, is to satisfy task dependencies within the tasks of the same MapReduce job; *all the Map tasks must finish before the execution of any Reduce task of the same job*. During the assignment and scheduling process, a number of different needs must be taken into account, e.g., transferring of the intermediate data (shuffle), data locality, and data skew, which give rise to the study of new scheduling problems.

Despite the importance and the challenging nature of scheduling in MapReduce environments, and despite the extensive investigation of a large variety of scheduling problems in parallel computing systems (see e.g., [13]), less attention has been paid to MapReduce scheduling problems. In fact, most of the previous work on scheduling in MapReduce systems concerns the experimental evaluation of scheduling heuristics, mostly from the viewpoint of finding good trade-offs between different objectives (see e.g., [14]). From a theoretical viewpoint, only few results on MapReduce scheduling have appeared so far [11, 4]. These are based on simplified abstractions of MapReduce scheduling, closely-related to some variants of the classical Open Shop and Flow Shop scheduling models, that capture issues such as task dependencies,

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

data locality, shuffle, and task assignment, under the key objective of minimizing the total weighted completion time of a set of MapReduce jobs.

In this direction, the theoretical model of Moseley et al. [11] generalizes a variant of the Flow Shop scheduling model, referred to as 2-stage Flexible Flow Shop (FFS), which is known to be strongly  $\mathcal{NP}$ -hard, even for jobs of a single Map and Reduce task and a single map and reduce processor (see in [11]). They consider the cases of both identical and unrelated processors and the goal is to minimize the total completion time of the jobs. For identical processors, they present a 12-approximation algorithm, and a  $O(1/\epsilon^2)$ -competitive online algorithm, for any  $\epsilon \in (0, 1)$ , under the assumption that the processors used by the online algorithm are  $1 + \epsilon$  times faster than the processors used by the optimal schedule. Since the identical processors setting fails to capture issues as data locality and to model communication costs between the Map and the Reduce tasks, Moseley et al. also consider the case of unrelated processors, which provides a more expressive theoretical model of scheduling in MapReduce environments. Nevertheless, they only consider the very restricted (and practically not so interesting) case where each job has a single Map and a single Reduce task, and present a 6-approximation algorithm and a  $O(1/\epsilon^5)$ -competitive online algorithm, for any  $\epsilon \in (0, 1)$ , under the assumption that the processors of the online algorithm are  $1 + \epsilon$  times faster.

A similar model of MapReduce scheduling so as to minimize the total completion time was proposed by Chen et al. [4]. In contrast with the model of [11], they assume that tasks are preassigned to processors and, in this restricted setting, they present an LP-based 8-approximation algorithm. Moreover, they deal with the shuffle phase in MapReduce systems and present a 58-approximation algorithm.

**Contribution and Results.** We adopt the theoretical model of [11] and consider MapReduce scheduling on unrelated processors. However, departing from [11], we deal with the general (and practically interesting) case where each job has any number of Map and Reduce tasks and we succeed in obtaining a polynomial-time constant approximation algorithm for minimizing the total weighted completion time. More specifically, we consider a set of MapReduce jobs to be executed on a set of unrelated processors. Each job consists of a set of Map tasks, that can be executed only on map processors, and a set of Reduce tasks, that can be executed only on Reduce processors. Each task has a different processing time for each processor and is associated with a positive weight, representing its importance. All jobs are available at time zero. Map or Reduce tasks can run simultaneously on different processors and, for each job, every Reduce task can start its execution after the completion of all the job's Map tasks. The goal is to find an assignment of the tasks to processors and schedule them *non-preemptively* so as to minimize their total weighted completion time.

In terms of classical scheduling, the model we consider in this work is a special case of total weighted completion time minimization on unrelated processors under precedence constraints. Despite its importance and generality, only few results are known for this problem. These results concern only the case of treelike precedence constraints [8]. More specifically, in [8], Kumar et al. propose a polylogarithmic approximation algorithm for the case where the undirected graph underlying the precedence constraints is a for-

est (a.k.a. treelike precedences). Their algorithm is based on a reduction from total weighted completion time minimization to an appropriate collection of makespan minimization problems. Based on ideas of [8], we present a  $(32 + \epsilon)$ -approximation algorithm for this problem that operates in two steps. In the first step, our algorithm computes a  $(8 + \epsilon)$ -approximation schedule for the Map tasks (resp. Reduce tasks) by combining a time indexed LP-relaxation of the problem with a well-known approximation algorithm for the makespan minimization problem on unrelated processors [9]. In fact, the makespan minimization algorithm runs on each time interval of the LP solution and computes an assignment of the Map (resp. Reduce) tasks to processors. In the second step, based on an idea from [11], we merge the two schedules, produced for the Map tasks and the Reduce tasks, into a single schedule that respects the precedence constraints. Using techniques from [11], we show that the merging step increases the approximation ratio by a factor of at most 4.

On the practical side, the theoretical model of [11] for MapReduce scheduling on unrelated processors deals with the most of the important aspects of the problem. So, considering jobs with any number of Map and Reduce tasks in this model is particularly important for practical applications, since the basic idea behind MapReduce computation is that each job is split into a large number of Map and Reduce tasks that can be executed in parallel (see e.g., [3, 6, 1, 2]). On the theoretical side, to the best of our knowledge, this is the first time that a constant approximation ratio is obtained for the problem of minimizing the total weighted completion time on unrelated processors under a nontrivial class of precedence constraints.

**Notation.** We consider a set  $\mathcal{J} = \{1, 2, \dots, n\}$  of  $n$  MapReduce jobs to be executed on a set  $\mathcal{P} = \{1, 2, \dots, m\}$  of  $m$  unrelated processors. Each job is available at time zero, is associated with a positive weight  $w_j$  and consists of a set  $\mathcal{M}$  of Map tasks and a set  $\mathcal{R}$  of Reduce tasks. Each task is denoted by  $\mathcal{T}_{k,j} \in \mathcal{M} \cup \mathcal{R}$ , where  $k \in N$  is the task index of job  $j \in \mathcal{J}$  and is associated with a vector of non-negative processing times  $\{p_{i,k,j}\}$ , one for each processor  $i \in \mathcal{P}_b$ , where  $b \in \{\mathcal{M}, \mathcal{R}\}$ . Let  $\mathcal{P}_{\mathcal{M}}$  and  $\mathcal{P}_{\mathcal{R}}$  be the sets of map and reduce processors respectively. Each job has at least one Map and one Reduce task that can run simultaneously on different processors and every Reduce task can start its execution after the completion of all Map tasks of the same job.

For a given schedule we denote by  $C_j$  and  $C_{k,j}$  the completion times of each job  $j \in \mathcal{J}$  and each task  $\mathcal{T}_{k,j} \in \mathcal{M} \cup \mathcal{R}$  respectively. Note that, due to the precedence constraints between Map and Reduce tasks,  $C_j = \max_{\mathcal{T}_{k,j} \in \mathcal{R}} \{C_{k,j}\}$ . By  $C_{max} = \max_{j \in \mathcal{J}} \{C_j\}$  we denote the makespan of the schedule, i.e., the completion time of the job which finishes last. Our goal is to schedule *non-preemptively* all Map tasks on processors of  $\mathcal{P}_{\mathcal{M}}$  and all Reduce tasks on processors of  $\mathcal{P}_{\mathcal{R}}$ , with respect to their precedence constraints, so as to minimize the total weighted completion time of the schedule, i.e.,  $\sum_{j \in \mathcal{J}} w_j C_j$ . We refer to this problem as MAPREDUCE SCHEDULING problem.

## 2. A CONSTANT APPROXIMATION ALGORITHM

In this section, we present a  $(32 + \epsilon)$ -approximation algorithm, for  $\epsilon \in (0, 1)$ , executed in the following two steps:

(i) it computes a  $(8 + \epsilon)$ -approximate schedule for assigning and scheduling all Map tasks (resp. Reduce tasks) on processors of the set  $\mathcal{P}_M$  (resp.  $\mathcal{P}_R$ ) and (ii) it merges the two schedules in one, with respect to the precedence constraints between Map and Reduce tasks of each job, increasing the approximation ratio by a factor of 4.

## 2.1 Scheduling Map and Reduce Tasks

Next, we propose an algorithm for the problem of minimizing the total weighted completion time of all Map (resp. Reduce) tasks on processors of the set  $\mathcal{P}_M$  (resp.  $\mathcal{P}_R$ ). For notational convenience, we use a dual variable  $b \in \{\mathcal{M}, \mathcal{R}\}$  to refer on either Map or Reduce sets of tasks.

We define  $(0, t_{\max} = \sum_{\mathcal{T}_{k,j} \in b} \max_{i \in \mathcal{P}_b} p_{i,k,j})$  to be the time horizon of potential completion times, where  $t_{\max}$  is an upper bound on the makespan of a feasible schedule. We discretize the time horizon into intervals  $(1, 1], (1, (1 + \delta)], ((1 + \epsilon), (1 + \delta)^2], \dots, ((1 + \delta)^{L-1}, (1 + \delta)^L]$ , where  $\delta \in (0, 1)$  is a small constant, and  $L$  is the smallest integer such that  $(1 + \delta)^{L-1} \geq t_{\max}$ . Let  $I_\ell = ((1 + \delta)^{\ell-1}, (1 + \delta)^\ell]$ , for  $0 \leq \ell \leq L$ , and  $\mathcal{L} = \{0, 1, 2, \dots, L\}$ . Note that, the number of intervals is polynomial in the size of the instance and to  $1/\delta$ . For each processor  $i \in \mathcal{P}_b$ , task  $T_{k,j} \in b$  and  $\ell \in \mathcal{L}$ , we introduce a variable  $y_{i,k,j,\ell}$  that denotes the fraction of task  $T_{k,j}$  assigned to processor  $i$  in time interval  $I_\ell$ . Furthermore, for each task  $T_{k,j} \in \mathcal{T}$ , we introduce a variable  $C_{k,j}$  corresponding to its completion time, and a variable  $z_{k,j}$  corresponding to its fractional processing time. For every job  $j \in \mathcal{J}$ , we also introduce a dummy task  $D_j$ , with zero processing time on every processor, which has to be processed after the completion of every other task  $T_{k,j} \in b$ .  $LP(b)$  is an interval-indexed linear programming relaxation of our problem.

$$LP(b) : \text{minimize } \sum_{j \in \mathcal{J}} w_j D_j$$

subject to :

$$\sum_{i \in \mathcal{P}_b, \ell \in \mathcal{L}} y_{i,k,j,\ell} = 1, \quad \forall \mathcal{T}_{k,j} \in b \quad (1)$$

$$z_{k,j} = \sum_{i \in \mathcal{P}_b} p_{i,k,j} \sum_{\ell \in \mathcal{L}} y_{i,k,j,\ell}, \quad \forall \mathcal{T}_{k,j} \in b \quad (2)$$

$$C_{D_j} \geq C_{k,j} + z_{k,j}, \quad \forall j \in \mathcal{J}, \mathcal{T}_{k,j} \in b \quad (3)$$

$$\sum_{i \in \mathcal{P}_b} \sum_{\ell \in \mathcal{L}} (1 + \delta)^{\ell-1} y_{i,k,j,\ell} \leq C_{k,j} \leq \sum_{i \in \mathcal{P}_b} \sum_{\ell \in \mathcal{L}} (1 + \delta)^\ell y_{i,k,j,\ell}, \quad \forall \mathcal{T}_{k,j} \in b \quad (4)$$

$$\sum_{\mathcal{T}_{k,j} \in b} p_{i,k,j} \sum_{t \leq \ell} y_{i,k,j,t} \leq (1 + \delta)^\ell, \quad \forall i \in \mathcal{P}_b, \ell \in \mathcal{L} \quad (5)$$

$$p_{i,k,j} > (1 + \delta)^\ell \Rightarrow y_{i,k,j,\ell} = 0, \quad \forall i \in \mathcal{P}_b, \mathcal{T}_{k,j} \in b, \ell \in \mathcal{L} \quad (6)$$

$$y_{i,k,j,\ell} \geq 0, \quad \forall i \in \mathcal{P}_b, \mathcal{T}_{k,j} \in b, \ell \in \mathcal{L} \quad (7)$$

Our objective is to minimize the sum of weighted completion times of all jobs. Constraint (1) ensures that each task is entirely assigned to processors of the set  $\mathcal{P}_b$  and constraint (2) defines its fractional processing time. Constraint (3) ensures that, for each job  $j \in \mathcal{J}$ , the completion of each task  $\mathcal{T}_{k,j}$  precedes the completion of task  $D_j$ . Constraint (4) adapts a lower and an upper bound on the completion time of each task. For each  $\ell \in \mathcal{L}$ , constraints (5) and (6) are validity constraints which state that the total fractional processing time on each processor is at most

$(1 + \delta)^\ell$ , and that if it takes time more than  $(1 + \delta)^\ell$  to process a task  $T_{j,k}$  on a processor  $i \in \mathcal{P}_b$ , then  $T_{k,j}$  should not be scheduled on  $i$ , respectively.

*Assignment and Scheduling.* Let  $(\bar{y}_{i,k,j,\ell}, \bar{z}_{k,j}, \bar{C}_{k,j})$  be an optimal (fractional) solution to  $LP(b)$ . For each  $2 \leq \ell \leq L$ , we define the set of tasks  $S(\ell) = \{T_{k,j} \in b \mid (1 + \delta)^{\ell-2}/2 \leq \bar{C}_{k,j} \leq (1 + \delta)^{\ell-1}/2\}$ , that complete their execution within the interval  $I_\ell$ . By definition, for each task  $\mathcal{T}_{k,j} \in S(\ell)$ , it must hold that  $2(1 + \delta)\bar{C}_{k,j} \leq (1 + \delta)^\ell$ .

We will assign all jobs of each set  $S(\ell)$  to processors in  $\mathcal{P}_b$  according to the following algorithm.

---

Algorithm MAKESPAN

- 1: Compute a basic feasible solution  $(\bar{x}_{i,k,j})$  to  $LP(T^*, b)$ .
  - 2: Assign all tasks having integral values to processors of  $\mathcal{P}_b$  as in  $(\bar{x}_{i,k,j})$ .
  - 3: Let a graph  $G = (A \cup \mathcal{P}_b, E)$ , where  $A = \{\mathcal{T}_{k,j} \mid 0 < x_{i,j,k} < 1\}$  and  $E = \{\{\mathcal{T}_{k,j}, i\} \mid \mathcal{T}_{k,j} \in A, i \in \mathcal{P}_b \text{ and } 0 < x_{i,k,j} < 1\}$ . Compute a perfect matching  $M$  on  $G$ .
  - 4: Assign each  $\mathcal{T}_{k,j} \in A$  to  $i \in \mathcal{P}_b$ , as indicated by  $M$ .
  - 5: **for** each assigned task  $\mathcal{T}_{k,j}$  **do**
  - 6:     Schedule  $\mathcal{T}_{k,j}$  as early as possible, non-preemptively, with processing time  $p_{i,k,j}$  on processor  $i \in \mathcal{P}_b$  that is assigned to. Let  $C_{k,j}$  be the completion time of  $\mathcal{T}_{k,j}$ .
- 

Algorithm MAKESPAN has been proposed in a seminal paper by Lenstra et al. [9] and it is based on the so-called parametric pruning technique in an LP setting. More specifically, if  $T$  is an estimation on the optimal makespan of a schedule of the jobs in  $S(\ell)$ , then by pruning away all task-processor pairs for which  $p_{i,k,j} > T$ , we are able to define a set of variables corresponding only to triples of the set  $\mathcal{Q}_T = \{(i, k, j) \mid p_{i,k,j} \leq T\}$ ; note that this pruning process has been already taken under consideration by constraints (6) of  $LP(b)$ . Since  $T \in \cup_{\ell' \leq \ell} I_{\ell'}$ , using binary search on  $\cup_{\ell' \leq \ell} I_{\ell'}$  with  $T$  as the search variable, we can find the minimum value of  $T$  such that the following system of linear constraints is feasible.

$$LP(b, T) :$$

$$\sum_{i:(i,k,j) \in \mathcal{Q}_T} x_{i,k,j} = 1 \quad \forall \mathcal{T}_{k,j} \in b \quad (8)$$

$$\sum_{\mathcal{T}_{k,j}:(i,k,j) \in \mathcal{Q}_T} x_{i,k,j} p_{i,k,j} \leq T \quad \forall i \in \mathcal{P}_b \quad (9)$$

$$x_{i,k,j} \geq 0 \quad \forall (i, k, j) \in \mathcal{Q}_T$$

Each variable  $x_{i,k,j}$  denotes the fractional processor assignment of each task  $\mathcal{T}_{k,j} \in S(\ell)$ . Now, if  $T^*$  is the minimum value for which  $LP(b, T)$  is feasible, then  $T^*$  is a lower bound on the optimal integral makespan.

Similarly as in [9], it can be proved that a basic feasible solution to  $LP(b, T)$  has at most  $|b| + |\mathcal{P}_b|$  non-zero variables, from which at least  $|b| - |\mathcal{P}_b|$ , must be set integrally. Then, the number of fractional  $x_{i,k,j}$  values must be at most  $2|\mathcal{P}_b|$ . If we formulate a bipartite graph  $G = (A \cup \mathcal{P}_b, E)$ , where  $A$  is the set of tasks having fractional  $x_{i,k,j}$  values and  $E = \{\{\mathcal{T}_{k,j}, i\} \mid \mathcal{T}_{k,j} \in A, i \in \mathcal{P}_b \text{ and } 0 < x_{i,k,j} < 1\}$ , then, according to the latter property, we deduce that  $G$  is a connected graph with at most  $2|\mathcal{P}_b|$  vertices and at most  $2|\mathcal{P}_b|$  edges. However, this means that  $G$  has the special topology of a pseudo-forest (a collection of trees with one possi-

ble extra edge) which enables the computation of a perfect matching on it. Hence, by executing steps 2-6 of Algorithm MAKESPAN, a non-preemptive schedule of tasks in  $S(\ell)$  can be found.

The following lemma provides a tight upper bound on the makespan of the schedule computed by Algorithm MAKESPAN.

LEMMA 1. *Algorithm MAKESPAN is a 2-approximation algorithm for scheduling the tasks of the set  $S(\ell)$  so as to minimize their makespan.*

In the next lemma, using filtering [10] we modify the  $y_{i,k,j,\ell}$  values of the solution to  $LP(b)$  to find an upper bound on the value of  $T^*$ .

LEMMA 2. *Consider a feasible solution to  $LP(b, T)$ . For each set of jobs  $S(\ell)$  that complete their execution within the interval  $I_\ell$ , it holds that  $T^* \leq 2(1 + \delta)^\ell$ , for  $\delta \in (0, 1)$ .*

As consequence of filtering in Lemma 2 the completion time of each task in  $S(\ell)$  is increased by a factor of 4; this result has already proven to be tight (see Section 2 in [12]).

---

Algorithm TASKSCHEDULING( $b$ )

- 1: Compute an optimal solution  $(\bar{y}_{i,k,j,\ell}, \bar{z}_{k,j}, \bar{C}_{k,j})$  to  $LP(b)$ .
  - 2: **for** each  $\ell \in \mathcal{L}$  **do**
  - 3:   compute  $S(\ell) = \{T_{k,j} \in b \mid (1 + \delta)^{\ell-2}/2 \leq \bar{C}_{k,j} \leq (1 + \delta)^{\ell-1}/2\}$
  - 4: **for** each  $\ell$  such that  $S(\ell) \neq \emptyset$  **do**
  - 5:   Schedule all tasks in  $S(\ell)$  by running Algorithm MAKESPAN.
- 

Running Algorithm TASKSCHEDULING( $b$ ), we compute a schedule for all Map (resp. Reduce) tasks such that:

THEOREM 1. *TASKSCHEDULING( $b$ ) is a  $(8 + \varepsilon)$ -approximation algorithm, for scheduling a set of Map (Reduce) tasks on a set of unrelated processors  $\mathcal{P}_M$  ( $\mathcal{P}_R$ ), in order to minimize their total weighted completion time, for  $\varepsilon \in (0, 1)$ .*

PROOF SKETCH. Let  $C_{k,j}$  be the completion time of a task  $T_{k,j} \in S(\ell)$ , in the schedule of Algorithm TASKSCHEDULING( $b$ ) and let  $C_{max}(\ell)$  be the makespan of the schedule of Algorithm MAKESPAN on the jobs in  $S(\ell)$ . Since,  $C_{k,j} \leq C_{max}(\ell)$ , for all  $T_{k,j} \in b$ , it suffices to prove that  $C_{k,j} \leq 8(1 + \delta)^2 \bar{C}_{k,j}$ : we combine Lemma 1 and Lemma 2 with the definition of the set  $S(\ell)$ . Then, as we can select an  $\varepsilon$  such that  $(1 + \delta)^2 \leq (1 + \varepsilon)$ , the theorem follows. Note that this ratio is tight.  $\square$

## 2.2 Merging Task Schedules

Let  $\sigma_M, \sigma_R$  be two schedules computed by two runs of Algorithm TASKSCHEDULING( $b$ ), for  $b = M$  and  $b = R$ , respectively. Let also  $C_j^{\sigma_M} = \max_{T_{j,k} \in M} \{C_{k,j}\}$ ,  $C_j^{\sigma_R} = \max_{T_{j,k} \in R} \{C_{k,j}\}$  be the completion times of the all Map and all Reduce tasks of a job  $j \in \mathcal{J}$  within these schedules, respectively. Depending on these completion time values, we assign each job  $j \in \mathcal{J}$  a width equal to  $\omega_j = \max\{C_j^{\sigma_M}, C_j^{\sigma_R}\}$ . The following algorithm computes a feasible schedule.

Algorithm MRS. In each time instant where a processor  $i \in \mathcal{P}_b$  becomes available, either it processes the Map task, assigned to  $i \in \mathcal{P}_M$  in  $\sigma_M$ , with the minimum width, or the

available (w.r.t. its precedence constraints) Reduce task, assigned to  $i \in \mathcal{P}_R$  in  $\sigma_R$ , with the minimum width.

By an analysis similar to that in [11], we can prove that:

THEOREM 2. *Algorithm MRS is a  $(32 + \varepsilon)$ -approximation for the MAPREDUCE SCHEDULING problem, for  $\varepsilon \in (0, 1)$ .*

PROOF SKETCH. By execution of Algorithm MRS, the feasibility of the resulted schedule can be easily verified. To prove the theorem, it suffices to prove that in such a schedule,  $\sigma$ , all tasks of a job  $j \in \mathcal{J}$  are completed by time  $2 \max\{C_j^{\sigma_M}, C_j^{\sigma_R}\}$ . Let  $C_j^\sigma$ , be the completion time of a job  $j \in \mathcal{J}$  in  $\sigma$ . Note that, for each of the Map tasks of  $j$ , their completion time is upper bounded by  $\omega_j$ . On the other hand, the completion time of each Reduce task is upper bounded by a quantity equal to  $r + \omega_j$ , where  $r$  is the earliest time when the task is available to be scheduled in  $\sigma$ . However,  $r = C_j^{\sigma_M} \leq \omega_j$  and thus  $C_j^\sigma \leq 2\omega_j = 2 \max\{C_j^{\sigma_M}, C_j^{\sigma_R}\}$ . By applying Theorem 1 and as we can select an  $\varepsilon$  such that  $\varepsilon \leq 4\varepsilon$ , the theorem follows.  $\square$

## 3. REFERENCES

- [1] F. Afrati, D. Fotakis, and J. Ullman. Enumerating subgraph instances using MapReduce. *IEEE-ICDE*: 62-73, 2013.
- [2] F. Afrati, A. D. Sarma, S. Salihoglu, and J. Ullman. Upper and Lower Bounds on the Cost of a MapReduce Computation. *VLDB*: 6(4):277-288, 2013.
- [3] F. Afrati and J. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE-TKDE*: 23(9):1282-1298, 2011.
- [4] F. Chen, M. S. Kodialam, and T. V. Lakshman. Joint scheduling of processing and shuffle phases in mapreduce systems. *INFOCOM*: 1143-1151, 2012.
- [5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI*: 137-150, 2004.
- [6] H. Karloff, S. Suri, and S. Vassilvitskii. A Model of Computation for MapReduce. *SODA*: 938-948, 2010.
- [7] R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in MapReduce and streaming. *ACM-SPAA*: 1-10, 2013.
- [8] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. Scheduling on unrelated machines under tree-like precedence constraints. *Algorithmica*: 55(1):205-226, 2009.
- [9] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*: 46:259-271, 1990.
- [10] J. Lin and J. S. Vitter. epsilon-approximations with minimum packing constraint violation. *SODA*: pages 771-782, 1992.
- [11] B. Moseley, A. Dasgupta, R. Kumar, and T. Sarlós. On scheduling in map-reduce and flow-shops. *ACM-SPAA*: 289-298, 2011.
- [12] J. R. Correa and M. Skutella and J. Verschae. The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders. *Math. Oper. Res.*: 379-398, 2012.
- [13] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2012.
- [14] D.-J. Yoo and K. M. Sim. A comparative review of job scheduling for mapreduce. *IEEE-ICIS*: 353-358, 2011.



# Binary Theta-Joins using MapReduce: Efficiency Analysis and Improvements

Ioannis K. Koumarelas  
Dept. of Informatics  
Aristotle University  
Thessaloniki, Greece  
koumarel@csd.auth.gr

Athanasios Naskos  
Dept. of Informatics  
Aristotle University  
Thessaloniki, Greece  
anaskos@csd.auth.gr

Anastasios Gounaris  
Dept. of Informatics  
Aristotle University  
Thessaloniki, Greece  
gounaria@csd.auth.gr

## ABSTRACT

We deal with binary theta-joins in a MapReduce environment, and we make two contributions. First, we show that the best known algorithm to date for this problem can reach the optimal trade-off between the size of the input a reducer can receive and the incurred communication cost when the join selectivity is high. Second, when the join selectivity is low, we present improvements upon the state-of-the-art with a view to decreasing the communication cost and the maximum load a reducer can receive, taking also into account the load imbalance across the reducers.

## 1. INTRODUCTION

Data analysis on voluminous data, such as clickstream data or data derived from scientific experiments and simulations, has given rise to the establishment of MapReduce as the most popular framework for large-scale processing. Analytical database queries remain a useful tool for big data analyses; however, such queries are being investigated in the MapReduce context rather than within a traditional DBMS environment. Analytical query processing in MapReduce has attracted a lot of interest, and the relevant work has investigated several issues, including indexing, data placement and layouts, optimizations, iterative processing, fair load allocation and interactive processing to name some of them [5]. In this work, we focus on improving the efficiency of join queries executed in MapReduce, for which several proposals already exist [7, 2, 9]. More specifically, we target binary theta-joins, where the join condition between two datasets is arbitrarily complex rather than a simple equation.

Nevertheless, most of the proposals to date tend to be developed on a best-effort basis, without systematically analyzing the inherent trade-offs. Two recent remedies to that have been proposed in [1, 8]. [8] introduces the notion of *minimal* MapReduce algorithms, which are algorithms accompanied by guarantees (up to a small constant) regarding several aspects, such as memory consumption and communication cost. The MapReduce rounds may be bounded but

they can be more than one. The work in [1] is complementary and presents a way to compute the lower bounds on communication cost as a function of the maximum input a reducer is allowed to receive for specific problems. This allows to define the trade-off between the load on the reducer side and the *replication rate*. The replication rate is defined as the average ratio of output to input key-value pairs on the map side, and is used as a metric of the communication cost. Further, the work in [1] examines whether known algorithms for those problems can match the lower bounds, provided that they consist of a single MapReduce round.

The algorithms 1-Bucket-Theta and M-Bucket in [7] form the basis of our work. Our first contribution is that we analyze the lower bounds for the binary theta-join problem and we show that the worst-case behaviour of 1-Bucket-Theta matches those bounds. However, such behaviour is expected only when the join selectivity is high. For low selectivities, and with the help of histograms, the more efficient M-Bucket-I and M-Bucket-O algorithms are presented in [7], which aim at minimizing the maximum reducer input and output, respectively. Our second contribution is that we enhance those algorithms through the clustering of histogram buckets. In that way, we can achieve more efficient partitioning of histogram buckets to reducers. The efficiency is measured in terms of the replication rate, the maximum reducer input, and the imbalance across reducers. We show that we can improve the replication rate (i.e., reduce the communication cost) and the maximum reducer input (i.e., reduce the longest running time and the space requirements of reducers) with insignificant impact on load imbalance.

The remainder of this extended abstract is structured as follows. In Sec. 2 we briefly present the 1-Bucket-Theta and M-Bucket algorithms, which we analyze in Sec. 3 and enhance in Sec. 4, respectively. In Sec. 5, we conclude and describe next steps.

## 2. BACKGROUND

In [7] the problem of performing binary theta joins  $S \bowtie_{\theta} T$  on MapReduce is studied. The core of the approach lies in how the workload is partitioned across reducers. To represent the workload, a *join matrix* ( $JM$ ) is used. In  $JMs$ , each cell corresponds to a pair of tuples, one from each input dataset, to be processed. The  $JM$  is split into several regions, where each region is mapped to a reducer. For each region, we can compute the amount of tuples that belong to it, which is the *input cost* of that region and is directly related to the computation and memory load of the associated reducer. For perfect load balancing, we want these regions to

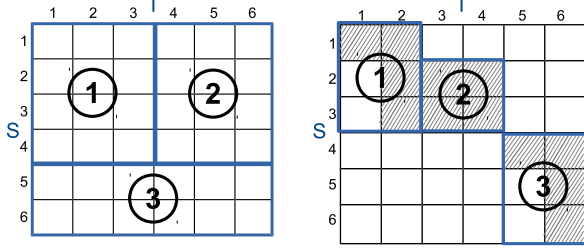


Figure 1: Partitioning the JM in 1-Bucket-Theta (left) and M-Bucket (right).

have equal input cost. In order to accomplish the latter objective, two main algorithms are presented: 1-Bucket-Theta and M-Bucket-I (and its variation M-Bucket-O).

## 2.1 1-Bucket-Theta

1-Bucket-Theta is the most generic algorithm, since it examines all tuple pairs (as in the Cartesian product), and requires minimal statistical information, namely just the cardinalities of the input. The strong point of the algorithm is the principled way that it partitions the JM, in a way that all JM cells are covered and, at the same time, the maximum reducer input is minimized. The algorithm is shown to be more suitable for high join selectivities (e.g., above 50%). Fig. 1(left) shows an example partitioning across 3 reducers, where there are 6 tuples from  $S$  and  $T$ , and the input cost of each reducer is 7 (4 tuples from  $S$  and 3 from  $T$ ), 7 (4 from  $S$  and 3 from  $T$ ) and 8 (2 from  $S$  and 6 from  $T$ ), respectively.

## 2.2 M-Bucket-I

In cases where there are histograms, so that we can safely reason as to whether a specific combination of tuples can satisfy the join condition, and the join selectivity is small, M-Bucket-I outperforms 1-Bucket-Theta. The histograms are equi-depth ones and are produced in a separate MapReduce phase, as explained in [7]. Then, the JM is constructed, where each cell corresponds to a pair of histogram buckets rather than a pair of tuples. As such, the size of a JM need not grow as the size of the input data increases at the expense of histogram buckets of higher depth. From the JM and the join condition, it is straightforward to identify pairs that do not contribute to the result (depicted as white cells in Fig. 1(right)). During the partitioning step, a heuristic method is followed, which is not accompanied by guarantees as in 1-Bucket-Theta but yields better results, since it benefits from the fact that most of the JM cells are not valid candidate pairs.

The difference between M-Bucket-I and M-Bucket-O is that the former targets the minimization of the maximum reducer input, whereas the latter targets the minimization of the maximum reducer output. Note that estimating the reducer output based on histograms is prone to significant errors, even when the histograms are accurate.

## 3. ON THE OPTIMALITY OF 1-BUCKET-THETA

First we define the lower bound on the communication cost of any 1-round MapReduce algorithm for binary theta-

joins. As already mentioned, the communication cost is measured using the replication rate metric. Let us examine the steps of the short version of the generic recipe for deriving such bounds from [1]. Given two relations  $S$  and  $T$ , with sizes  $|S|$  and  $|T|$ , respectively, we have:

- **Size (Number) of Inputs and Outputs:**

- Inputs:  $|S|+|T|$
- Outputs:  $|S||T|$  (accounting for the worst case, which is the cartesian product)

- **Deriving  $g(q)$ :** The upper bound of outputs a reducer can produce given  $q$  inputs, denoted as  $g(q)$ , occurs when  $q$  is equally divided into input from  $S$  and  $T$ , i.e.,  $\frac{q}{2}$  tuples from  $S$  and  $\frac{q}{2}$  tuples from  $T$ . The maximum result of applying the theta join on these two quantities is when we have a cartesian product, thus  $g(q) = \frac{q^2}{4}$ .

- **Replication Rate  $r(q)$ :** The quantity  $\frac{g(q)}{q}$  equals  $\frac{q^2}{4q} = \frac{q}{4}$ , which is monotonically increasing in  $q$ . Therefore, the replication rate can be computed using the formula:  $r(q) \geq \frac{q|O|}{g(q)I} = \frac{4(|S||T|)}{q(|S|+|T|)}$ . So, the lower bound on  $r$ ,  $r_{lb}$ , is  $\frac{4(|S||T|)}{q(|S|+|T|)}$ .

The above formula illustrates the exact trade-off between parallelism and communication cost in binary theta-joins. By increasing the degree of parallelism in order to decrease the input  $q$  each reducer receives, the communication cost increases, since, for the lower bound,  $q$  and  $r$  are inversely proportional to each other.

The next step is to find the upper bound on replication rate of 1-Bucket-Theta. In [7], three partitioning cases are presented, based on the sizes  $|S|$  and  $|T|$  and the number of available reducer processors  $p$ . Due to the limited space, we will examine only the first case in detail.

The first case corresponds to the scenario, where the JM can be exactly covered by  $c_S \times c_T$  squares of side-length  $\sqrt{|S||T|/p}$ . This means that the following conditions hold:  $|S| = c_S \sqrt{|S||T|/p}$  and  $|T| = c_T \sqrt{|S||T|/p}$ , where  $c_S, c_T$  are positive integers. For example, if  $p = 4$ , then the JM in Fig. 1(left) can be exactly covered by 4 squares of side-length 3. Then we have:

- Replication rate of 1-Bucket-Theta ( $r_{1BT}$ ):

$$r_{1BT} \leq \frac{|S|c_T+|T|c_S}{|S|+|T|} = \frac{\frac{|S||T|}{p} + \frac{|T||S|}{p}}{\frac{|S||T|}{p}} = \frac{2|S||T|}{(\sqrt{\frac{|S||T|}{p}})(|S|+|T|)}$$

- Reducer input:  $q_{1BT} = 2\sqrt{\frac{|S||T|}{p}}$

- Combining  $r_{1BT}$  and  $q_{1BT}$ :

$$r_{1BT}q_{1BT} \leq \left(\frac{2|S||T|}{(\sqrt{\frac{|S||T|}{p}})(|S|+|T|)}\right)(2\sqrt{\frac{|S||T|}{p}}) = 4\frac{|S||T|}{|S|+|T|}$$

which implies that  $r_{1BT}(q_{1BT}) \leq r_{lb}$

So, the upper bound of the first case of 1-Bucket-Theta is at most as high as the lower bound of the problem, which means that, for that case, the algorithm is optimal.

Following the same reasoning, the other two cases (Theorems 2 and 3 in [7], respectively), which correspond to different formulas for  $c_S$  and  $c_T$ , can be examined, for which we have:

- Case 2:  $r_{1BT} \leq \frac{4}{q} \frac{|T||S|}{|S|+|T|} = r_{lb}$
- Case 3:  $r_{1BT} \leq \frac{8}{q} \frac{|S||T|}{|S|+|T|} = 2r_{lb}$

Overall, the upper bound of the replication rate is at most two times the lower bound, and as such is optimal up to a constant factor. In [3], it is shown that the lower bound can be met for self-joins, which is special case of binary joins.

## 4. REDUCING THE REPLICATION RATE IN M-BUCKET

The partitioner of M-Bucket-I algorithm operates on a join matrix (JM), where each cell corresponds to a pair of histogram buckets. It tries to fit the cells in rectangular regions; each region is associated with a single reducer. The rationale of our approach is to permute JM’s rows and columns, in order to improve the quality of the partitioning phase.

The problem of cell rearrangement can be addressed with several algorithm families, such as *clustering* (e.g., hierarchical, array-based, and so on), *combinatorial optimization* (e.g., bin packing, knapsack) and *bandwidth reduction*. Here, we examine the impact of array-based clustering algorithms and more specifically, we employ the Bond Energy clustering algorithm (BEA) [6], due to its efficiency [4]. The purpose of BEA is to identify natural clusters that occur in complex data arrays, such as JMs. This task is accomplished by permuting the rows and columns of the JM in a way that the numerically larger array elements are clustered together. As the JM of our interest comprises a two-dimensional bitmap array, i.e. the cell values are either 0 or 1 to indicate whether the processing of the corresponding pairs is meaningful or not, we expect all the non-zero values to be grouped as close as possible. The intuition is that, if the JM contains more empty sub-matrices, the mapping of the remainder sub-matrices to reducers will improve.

Our work adds a step of beforehand analysis to the M-Bucket-I/O algorithm, just after the histograms are built and the initial JM is produced. It thus takes place before the actual execution on a MapReduce platform. The quality of a JM is assessed with the help of the following three metrics:

1. *replication rate (rep)*, defined as in the Introduction and [1];
2. *maximum reducer input (mri)*; and
3. *input imbalance (imb)*, defined as the ratio of *mri* to the average reducer input, considering only the non-idle reducers.

Note that the metrics above can be accurately computed from the JM, without requiring the real execution to be completed. Thus, if the JM rearrangement is considered as not beneficial, the execution can switch back to the original JM. That is, it is straightforward to add a post-processing phase, in order to guarantee that we choose the best partitioning between the one based on the original and the one based on the re-arranged JM. Consequently, our proposal does never lead to performance degradation; actually it can lead to significant improvements according to our experiments.

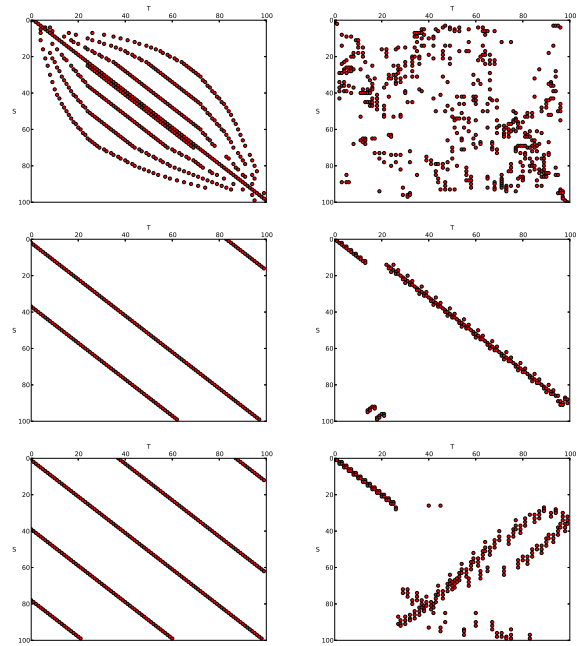


Figure 2: Example JMs before (left) and after (right) applying BEA.

As an example, we extracted a sample of 64M tuples from the Cloud dataset in <http://cdiac.ornl.gov/ftp/ndp026c/ndp026c.pdf>. Fig. 2(top) shows the initial and rearranged JM for a self-join query that retrieves record pairs, for which the absolute difference of the sea level is between 0 and 2, or between 22 and 24, or between 50 and 52, or between 80 and 82 to give an example of a complex range query. The rearranged JM yields 21% lower *rep* and 19% lower *mbi* at the expense of 4% higher *imb*. Next, we proceed to more systematic experiments on synthetic data.

### 4.1 Experimental Evaluation

We focus on band joins, which is a type of theta-joins that can significantly benefit from M-Bucket. In band joins, the condition is in the form of  $R.A - \epsilon \leq S.A \leq R.A + \epsilon$ . The experimental setup is as follows. We randomly generate synthetic JMs so that the produced JMs vary in the following aspects: join selectivity, number of band conditions, and size of JMs. Then, we compute the statistics of the resulting partitioning to reducers both when we cluster the JM and when we do not. In the first experiment, we assume that the dimensions of the JM are  $100 \times 100$ . We vary the number of available reducers from 10 to 40. Also, the numbers of band conditions examined are 1, 3 and 5. For each band condition, we examined selectivity values of 1%, 5% and 10%.

Fig. 2 shows two more examples of JM rearrangement. From the left column of the middle and bottom row, we can see the typical form of the original synthetic JMs. For each band condition, there is a diagonal stripe of cells, for which the join condition holds. The gaps between such stripes are randomly shifted, so that the JMs are not symmetric; for each condition the selectivity is set to 1%. As we can observe, the effect of the BEA algorithm is optically widely different, but in both cases, there were significant improvements, which we discuss below.

The average impact of BEA on the metrics examined are

	<i>rep</i>	<i>mri</i>	<i>imb</i>	coverage
Overall	0.846	0.880	1.029	59.26%
Band Selectivity				
1%	0.717	0.735	1.028	66.67%
5%	0.920	0.949	1.014	66.67%
10%	0.928	0.996	1.056	44.45%
Number of Band Conditions				
1	0.987	0.967	0.964	33.34%
3	0.821	0.835	1.010	44.45%
5	0.810	0.873	1.058	100%

Table 1: Average ratio of the BEA-produced JM metrics to the original JM metrics.

	<i>rep</i>	<i>mri</i>	<i>imb</i>
Overall	0.634	0.649	1.023
Band Selectivity			
1%	0.634	0.649	1.023
5%	0.833	0.875	1.050
10%	0.848	0.900	1.050
Number of Band Conditions			
1	0.979	1	0.988
3	0.737	0.733	0.995
5	0.634	0.649	1.023

Table 2: Ratio of the BEA-produced JM metrics to the original JM metrics for the maximum *rep* drop observed.

summarized in Table 1. The rightmost column of the table shows the percentage of the times that the rearranged JM has led to improvements in the replication rate. Table 2 refers to the maximum improvements regarding replication observed. From these two tables, we can draw the following conclusions. On average, our proposal improves the partitioning in approximately 59% of the times. In those cases, the average decrease in the replication rate is 15%, but it can reach 37%. The improvements become more significant as the number of the band conditions increase and the selectivity becomes lower. On average, when the band selectivity is 1%, the replication rate drops by 28%, while the maximum reducer input decreases by 26%. There is a slight increase in the relative imbalance though. Similarly, we can observe, that, when the number of band conditions is 5, there are improvements in all the cases examined.

We also investigated the impact of the number of reducers, but this was not found to be significant. Finally, note that we considered only the cases where the replication rate is strictly less than that with the original JMs in order to compute *mri* and *imb*. The average values of these two metrics are slightly different if all the measurements are considered.

We conducted an additional experiment, where we increased the dimensions of the JM to  $1000 \times 1000$  and we further decreased the minimum selectivity of each band condition to 0.1%. The main purpose was to verify our hypothesis that our proposal is more suitable for band joins with multiple conditions, each having a low selectivity. Indeed, in 100% of the cases examined when the selectivity was 0.1% and the number of band conditions was 3 and 5, there was a significant decrease in the replication rate (28.1% on average). The maximum reducer input was also decreased by the same amount, whereas the imbalance remained similar. Overall, when the selectivity is low, there is more space for BEA to yield empty sub-matrices; whereas, when there are fewer band conditions, the differences from the original JMs are less significant.

## 5. CONCLUSIONS AND FURTHER WORK

We investigate the execution of binary theta-joins using MapReduce. First we analyze the efficiency of the state-of-the-art and second, we propose the usage of a pre-processing clustering algorithm in order to help the partitioning of the map output to reducers. Our proposal was shown to incur significant reductions in the communication cost and the maximum input received by each reducer when the theta clause comprises several conditions, each of low selectivity. A strong point of our approach is that it is not intrusive, in the sense that it can be easily incorporated into the current state-of-the-art proposal in [7], as a pre-processing phase before the actual execution on a MapReduce platform begins. In addition, it is straightforward to assess whether our approach is beneficial for a specific setting, and thus our proposal does not lead to overall performance degradation.

In the future, we plan to focus on more elaborate types of array rearrangement algorithms. Scalability is also an issue, since algorithms such as BEA do not scale to matrices with very large dimensions. Another avenue for further work is to investigate more sophisticated partitioning algorithms to be coupled with JM rearrangement. Harder problems include the investigation of provably optimal techniques for multi-way theta-joins and efficient histogram construction when there are multiple attributes participating in the theta-join condition.

*Acknowledgments* This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

## 6. REFERENCES

- [1] F. N. Afrati, A. D. Sarma, S. Salihoglu, and J. D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.
- [2] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.
- [3] F. N. Afrati and J. D. Ullman. Matching bounds for the all-pairs mapreduce problem. In *IDEAS*, pages 3–4, 2013.
- [4] S. Climer and W. Zhang. Rearrangement clustering: Pitfalls, remedies, and applications. *Journal of Machine Learning Research*, 7:919–943, 2006.
- [5] C. Doukeridis and K. Nørnvåg. A survey of large-scale analytical query processing in mapreduce. *The VLDB Journal*, pages 1–26, 2013.
- [6] W. T. McCormick, P. J. Schweitzer, and T. W. White. Problem decomposition and data reorganization by a clustering technique. *Operations Research*, 20(5):993–1009, 1972.
- [7] A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. In *SIGMOD*, pages 949–960, 2011.
- [8] Y. Tao, W. Lin, and X. Xiao. Minimal mapreduce algorithms. In *SIGMOD*, pages 529–540, 2013.
- [9] X. Zhang, L. Chen, and M. Wang. Efficient multi-way theta-join processing using mapreduce. *PVLDB*, 5(11):1184–1195, 2012.

# On the design space of MapReduce ROLLUP aggregates

Duy-Hung Phan  
EURECOM  
phan@eurecom.fr

Matteo Dell’Amico  
EURECOM  
dellamic@eurecom.fr

Pietro Michiardi  
EURECOM  
michiard@eurecom.fr

## ABSTRACT

We define and explore the design space of efficient algorithms to compute ROLLUP aggregates, using the MapReduce programming paradigm. Using a modeling approach, we explain the non-trivial trade-off that exists between parallelism and communication costs that is inherent to a MapReduce implementation of ROLLUP. Furthermore, we design a new family of algorithms that, through a single parameter, allow to find a “sweet spot” in the parallelism vs. communication cost trade-off. We complement our work with an experimental approach, wherein we overcome some limitations of the model we use. Our results indicate that efficient ROLLUP aggregates require striking the good balance between parallelism and communication for both one-round and chained algorithms.

## 1. INTRODUCTION

Online analytical processing (OLAP) is a fundamental approach to study multi-dimensional data involving the computation of, for example, aggregates on data that are accumulated in traditional data warehouses. When operating on massive amounts of data, it is typical for business intelligence and reporting applications, to require data summarization, which is achieved using standard SQL operators such as GROUP BY, ROLLUP, CUBE, and GROUPING SETS.

Despite the tremendous amount of work carried out in the database community to come up with efficient ways of computing data aggregates, little work has been done to extend these lines of work to cope with massive scale. Indeed, the main focus of prior works in this domain has been on single server systems or small clusters executing a distributed database, implementing efficient implementations of CUBE and ROLLUP operators, in line with the expectations of low-latency access to data summaries [6, 8, 11, 13, 14, 19]. Only recently, the community devoted attention to solve the problem of computing data aggregates at massive scales using data intensive, scalable computing engines such

as MapReduce [10]. In support of the growing interest in computing data aggregates on batch-oriented systems, several high-level languages built on top of MapReduce, such as PIG [3] and HIVE [2], support simple implementations of, for example, the ROLLUP operator.

The endeavor of this work is to take a systematic approach to study the design space of the ROLLUP operator: besides being widely used on its own, ROLLUP is also a fundamental building block used to compute CUBE and GROUPING SETS [7]. We study the problem of defining the design space of algorithms to implement ROLLUP through the lenses of a recent model of MapReduce-like systems [4]. The model explains the trade-offs that exist between the degree of parallelism that is possible to achieve and the communication costs that are inherently present when using the MapReduce programming model. In addition, we overcome current limitations of the model we use (which glosses over important aspects of MapReduce computations) by extending our analysis with an experimental approach. We present instances of algorithmic variants of the ROLLUP operator that cover several points in the design space, implement and evaluate them using an Hadoop cluster.

In summary, our contributions are the following:

- We study the design space that exists to implement ROLLUP and show that, while it may appear deceptively simple, it is not a straightforward embarrassing parallel problem. We use modeling to obtain bounds on parallelism and communication costs.
- We design and implement new ROLLUP algorithms that can match the bounds we derived, and that swipe the design space we were able to define.
- We pinpoint the essential role of *combiners* (an optimization allowing pre-aggregation of data, which is available in real instances of the MapReduce paradigm, such as Hadoop [1]) for the practical relevance of some algorithm instances, and proceed with an experimental evaluation of several variants of ROLLUP implementations, both in terms of their performance (runtime) and their efficient use of cluster resources (total amount of work).
- Finally, our ROLLUP implementations exist in Java MapReduce and have been integrated in our experimental branch of PIG, which are available in a public repository.<sup>1</sup>

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

<sup>1</sup><https://bitbucket.org/bigfootproject/rollupmr>

The remainder of this paper is organized as follows. Section 2 provides background information on the model we use in our work and presents related work. Section 3 illustrates a formal problem statement and Section 4 presents several variants of ROLLUP algorithms. Section 5 outlines our experimental results to evaluate the performance of the algorithms we introduce in this work. Finally, Section 6 concludes our work and outlines future research directions.

## 2. BACKGROUND AND RELATED WORK

We assume the reader to be familiar with the MapReduce [10] paradigm and its open-source implementation Hadoop [1, 20]. First, we give a brief summary of the model introduced by Afrati *et al.* [4], which is the underlying tool we use throughout our paper. Then, we present related works that focus on the MapReduce implementation of popular data analytics algorithms.

**The MapReduce model.** Afrati *et al.* [4] recently studied the MapReduce programming paradigm through the lenses of an original model that elucidates the trade-off between parallelism and communication costs of single-round MapReduce jobs. The model defines the design space of a MapReduce algorithm in terms of *replication rate* and *reducer-key size*. The replication rate  $r$  is the average number of (key, value) pairs created from each input in the map phase, and represents the communication cost of a job. The reducer-key size  $q$  is the upper bound of the size of list of values associated to a reducer-key. Jobs have higher degrees of parallelism when  $q$  is smaller. For some problems, parallelism comes at the expense of larger communication costs, which may dominate the overall execution time of a job.

Afrati *et al.* show how to determine the relation between  $r$  and  $q$ . This is done by first bounding the amount of input a reducer requires to cover its outputs. Once this relation is established, a simple yet effective “recipe” can be used to relate the size of the input of a job to the replication rate and to the bounds on output covering introduced above. As a consequence, given a problem (*e.g.*, finding the Hamming distance between input strings), the model can be used to establish bounds on  $r$  and  $q$ , which in turn define the design space that instances of algorithms solving the original problem may cover.

**Related work.** Designing efficient MapReduce algorithms to implement a wide range of operations on data has received considerable attention recently. Due to space limitations, we cannot give justice to all works that addressed the design, analysis and implementation of graph algorithms, clustering algorithms and many other important problems: here we shall focus on algorithms to implement SQL-like operators. For example, the relational JOIN operator is not supported directly in MapReduce. Hence, attempts to implement efficient JOIN algorithms in MapReduce have flourished in the literature: Blanas *et al.* [9] studied *Repartition Join*, *Broad-cast Join*, and *Semi-Join*. More recent work tackle more general cases like theta-joins [17] and multi-way-joins [5].

With respect to OLAP data analysis tasks such as CUBE and ROLLUP, efficient MapReduce algorithms have only lately received some attention. A first approach to study CUBE and ROLLUP aggregates has been proposed by Nandi *et al.* [16]; this algorithm, called “naive” by the authors, is called *Vanilla* in this work. MR-Cube [16] mainly focuses on

algebraic aggregation functions, and deals with data skew; it implements the CUBE operator by breaking the algorithm in three phases. A first job samples the input data to recognize possible reducer-unfriendly regions; a second job breaks those regions into sub-regions, and generates corresponding (key, value) pairs to all regions, to perform partial data aggregation. Finally, a last job reconstructs all sub-regions results to form the complete output. The MR-Cube operator naturally implements ROLLUP aggregates. However in the special case of ROLLUP, the approach has two major drawbacks: it replicates records in the map phase as in the naive approach and it performs redundant computation in the reduce phase.

For the sake of completeness, we note that one key idea of our work (in-reducer grouping) shares similar traits to what is implemented in the Oracle database [7]. However, the architectural differences with respect to a MapReduce system like Hadoop, and our quest to explore the design space and trade-offs of ROLLUP aggregates make such work complementary to ours.

## 3. PROBLEM STATEMENT

We now define the ROLLUP operation as a generalization of the SQL ROLLUP clause, introducing it by way of a running example. We use the same example in Section 4 to elucidate the details of design choices and, in Section 5, to benchmark our results.

ROLLUP can be thought of as a hierarchical GROUP BY at various granularities, where the grouping keys at a coarser granularities are a subset of the keys at a finer granularity. More formally, we define the ROLLUP operation on an input *data set*, an *aggregation function*, and a set of  $n$  *hierarchical granularities*:

- We consider a *data set* akin to a database table, with  $M$  columns  $c_1, \dots, c_M$  and  $L$  rows  $r_1, \dots, r_L$  such that each row  $r_i$  corresponds to the  $(r_{i1}, \dots, r_{iM})$  tuple.
- Given a set of rows  $R \subseteq \{r_1, \dots, r_L\}$ , an *aggregation function*  $f(R)$  produces our desired result.
- $n$  *granularities*  $d_1, \dots, d_n$  determine the groupings that an input data is subject to. Each  $d_i$  is a subset of  $\{c_1, \dots, c_M\}$ , and granularities are *hierarchical* in the sense that  $d_i \subsetneq d_{i+1}$  for each  $i \in [1, n - 1]$ .

The ROLLUP computation returns the result of applying  $f$  after grouping the input by the set of columns in each granularity. Hence, the output is a new table with tuples corresponding to grouping over the finest ( $d_n$ ) up to the coarsest ( $d_1$ ) granularity, denoting irrelevant columns with an ALL value [12].

**Example.** Consider an Internet Service provider which needs to compute aggregate traffic load in its network, per day, month, year and in overall. We assume input data to be a large table with columns  $(c_1, c_2, c_3, c_4)$  corresponding to (year, month, day, payload<sup>2</sup>). A few example records from this dataset are shown in the following:

```
(2012, 3, 14, 1)
(2012, 12, 5, 2)
(2012, 12, 30, 3)
(2013, 5, 24, 4)
```

<sup>2</sup>In Kilobytes

The aggregation function  $f$  outputs the sum of values over the  $c_4$  (payload) column. Besides SUM, other typical aggregation functions are MIN, MAX, AVG and COUNT; it is also possible to consider aggregation functions that evaluate data in multiple columns, such as for example correlation between values in different columns.

Input granularities are  $d_1 = \emptyset$ ,  $d_2 = \{\text{year}\}$ ,  $d_3 = \{\text{year}, \text{month}\}$ , and  $d_4 = \{\text{year}, \text{month}, \text{day}\}$ . The highest granularity,  $d_1 = \emptyset$ , groups on no columns and is therefore equivalent to a SQL GROUP BY ALL clause that computes the overall sum of the payload column; such an overall aggregation is always computed in SQL implementations, but it is not required in our more general formulation. We will see in the following that “global” aggregation is problematic in MapReduce.

In addition to aggregation on hierarchical time periods as in this case, ROLLUP aggregation applies naturally to other cases where data can be organized in tree-shaped taxonomies, such as for example country-state-region or unit-department-employee hierarchies.

If applied on the example, the ROLLUP operation yields the following result (we use ‘\*’ to denote ALL values):

```
(2012, 3, 14, 1)
(2012, 3, *, 1)
(2012, 12, 5, 2)
(2012, 12, 30, 3)
(2012, 12, *, 5)
(2012, *, *, 6)
(2013, 5, 24, 4)
(2013, 5, *, 4)
(2013, *, *, 4)
(*, *, *, 10)
```

Rows with ALL values represent the result of aggregation at coarser granularities: for example, the (2012, \*, \*, 6) tuple is the output of aggregating all tuples from year 2012.

**Aggregation Functions and Combiners.** In MapReduce, it is possible to pre-aggregate values computed in mappers by defining *combiners*. We will see in the following that combiners are crucial for the performance of algorithms defined in MapReduce. While many useful aggregation functions are susceptible to being optimized through combiners, not all of them are. Based on the definition by Gray *et al.* [12], when an aggregation function is *holistic* there is no constant bound on the size of a combiner output; representative holistic functions are MEDIAN, MODE and RANK.

The algorithms we define are differently susceptible to the presence and effectiveness of combiners. When discussing the merits of each implementation, we also consider the case where aggregation functions are holistic and hence combiners are of little or no use.

## 4. THE DESIGN SPACE

We explore the design space of ROLLUP, with emphasis on the trade-off between communication cost and parallelism. We first apply a model to obtain theoretical bounds on replication rate and reducer key size; we then consider two algorithms (Vanilla and In-Reducer Grouping) that are at the end-points of the aforementioned trade-off, having respectively maximal parallelism and minimal communication cost. We follow up by proposing various algorithms that

operate in different, and arguably more desirable, points of the trade-off space.

### 4.1 Bounds on Replication and Parallelism

Here we adopt the model by Afrati *et al.* [4] to find upper and lower bounds for the replication rate. Note that the model, unfortunately, does not account for combiners nor for multi-round MapReduce algorithms.

First, we define the number of all possible inputs and outputs to our problem, and a function  $g(q)$  that allows to evaluate the number of outputs that can be covered with  $i$  input records. To do this, we refer to the definitions in Section 3:

1. **Input set:** we call  $C_i$  the number of different values that each column  $c_i$  can take. The total number of inputs is therefore  $|I| = \prod_{i=1}^M C_i$ .
2. **Output set:** for each granularity  $d_i$ , we denote the number of possible grouping keys as  $N_i = \prod_{C_i \in d_i} C_i$  and the number of possible values that the aggregation function can output as  $A_i$ .<sup>3</sup> Thus, the total number of outputs is  $|O| = \sum_{i=1}^n N_i A_i$ .
3. **Covering function:** let us consider a reducer that receives  $q$  input records. For each granularity  $d_i$ , there are  $N_i$  grouping keys, each one grouping  $|I|/N_i$  inputs and producing  $A_i$  outputs. The number of groups that the reducer can cover at granularity  $d_i$  is therefore no more than  $\lfloor qN_i/|I| \rfloor$ , and the covering function is  $g(q) = \sum_{i=1}^n A_i \lfloor \frac{qN_i}{|I|} \rfloor$ .

**Lower Bound on Replication Rate.** We consider  $p$  reducers, each receiving  $q_i \leq q$  inputs and covering  $g(q_i)$  outputs. Since together they must cover all outputs, it must be the case that  $\sum_{j=1}^p g(q_j) \geq |O|$ . This corresponds to

$$\sum_{j=1}^p \sum_{i=1}^n A_i \left\lfloor \frac{q_j N_i}{|I|} \right\rfloor \geq \sum_{i=1}^n N_i A_i. \quad (1)$$

Since  $q_j N_i/|I| \geq \lfloor q_j N_i/|I| \rfloor$ , we obtain the lower bound of the replication rate  $r$  as:

$$r = \sum_{i=1}^p \frac{q_i}{|I|} \geq 1. \quad (2)$$

Equation 2 seems to imply that ROLLUP aggregates is an *embarrassingly parallel* problem: the  $r \geq 1$  bound on replication rate does not depend on the size  $q_i$  of reducers. In Section 4, we show – for the first time – an instance of an algorithm that matches the lower bound. Instead, known instances of ROLLUP aggregates have a larger replication rate, as we shall see next.

**Limits on Parallelism.** Let us now reformulate Equation 2, this time requiring only that the output of the coarsest granularity  $d_1$  is covered. We obtain

$$\sum_{j=1}^p \left\lfloor \frac{q_j N_1}{|I|} \right\rfloor \geq N_1.$$

<sup>3</sup>For the limit case  $d_i = \emptyset$ ,  $N_i = 1$ , corresponding to the single empty grouping key.



Clearly, the output *cannot be covered* (the left side of the equation would be zero) unless there are reducers receiving at least  $q_j \geq |I|/N_1$  input records. Indeed, the coarsest granularity imposes hard limits on the parallelism, requiring to broadcast the full input on at most  $N_1$  reducers. This is exacerbated if – as it is the case with the standard SQL ROLLUP – there is an overall aggregation, resulting in  $d_1 = \emptyset$ ,  $N_1 = 1$  and therefore  $q_j \geq |I|$ . A *single reducer* needs to receive *all the input*: it appears that no parallelism whatsoever is possible.

As we show in the following, this negative result however depends on the limitations of the model: by applying combiners and/or multiple rounds of MapReduce computation, it is indeed possible to compute efficient ROLLUP aggregates in parallel.

**Maximum Achievable Parallelism.** Our model considers parallelism as determined by the number of reducers  $p$  and the number of input records  $q_j$  each of them processes. However, one may also consider the number of *output* records produced by each reducer: in that case, the maximum parallelism achievable is when each reducer produces at most a single output value. This can be obtained by assigning each grouping key in each granularity to a different reducer; the aggregation function is then guaranteed to output only one of the  $A_i$  possible values. This, however, implies a replication rate  $r = n$ ; an implementation of the idea is described in the following section.

## 4.2 Baseline algorithms

Next, we define two baseline algorithms to compute ROLLUP aggregates: Vanilla, which is discussed in [16], and *In-Reducer Grouping*, which is our contribution. Then, we propose a hybrid approach that combines both baseline techniques.

**Vanilla Approach.** We describe here an approach that maximizes parallelism at the detriment of communication cost; since this is the approach which is currently implemented in Apache Pig [15] we refer to it as Vanilla. Nandi *et al.* [16] refer to it as “naive”.

The ROLLUP operator can be considered as the result multiple GROUP BY operations: each of them is carried out at a different granularity. Thus, to perform ROLLUP on  $n$  granularities, for each record, the vanilla approach generates exactly  $n$  records corresponding to these  $n$  grouping sets (each grouping sets belongs to one granularity). For instance, taking as input the (2012, 3, 14, 1) record of the running example, this approach generates 4 records as outputs of the map phase:

```
(2012, 3, 14, 1) (day granularity)
(2012, 3, *, 1) (month granularity)
(2012, *, *, 1) (year granularity)
(*, *, *, 1) (overall granularity)
```

The Reduce step performs exactly as the reduce step of a GROUP BY operation, using the first three records (year, month, day) as keys. By doing this, reducers pull all the data that is needed to generate each output record (shuffle step), and compute the aggregate (reduce step). Figure 1 illustrates a walk-through example of the vanilla approach with just 2 records.

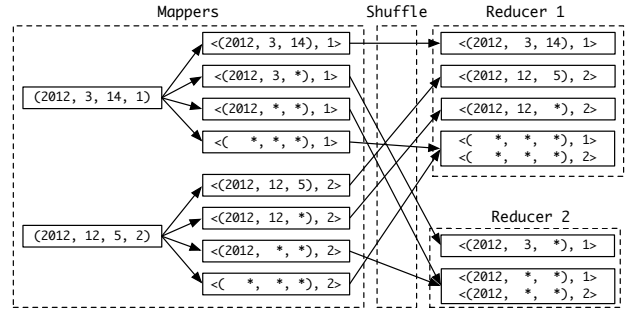


Figure 1: Example for the vanilla approach.

*Parallelism and Communication Cost.* The final result of ROLLUP is computed in a single MapReduce job. As discussed above, this implementation obtains the maximum possible degree of parallelism, since it can be parallelized up to a level where a single reducer is responsible of a single output value. On the other hand, this algorithm requires maximal communication costs, since for each input record,  $n$  map output records are generated. In addition, when the aggregation operation is *algebraic* [12], redundant computation is carried out in the reduce phase, since results computed for finer granularities cannot be reused for the coarser ones.

*Impact of Combiners.* This approach largely benefits from combiners whenever they are available, since they can compact the output computed at the coarser granularity (*e.g.*, in the example the combiner is likely to compute a single per-group value at the *year* and overall granularity). Without combiners, a granularity such as overall would result in shuffling data from *every input tuple* to a *single reducer*.

While combiners are very important to limit the amount of data sent along the network, the large amount of temporary data generated with this approach is still problematic: map output tuples need to be buffered in memory, sorted, and eventually spilled to disk if the amount of generated data does not fit in memory. This results, as we show in Section 5, in performance costs that are discernible even when combiners are present.

**In-Reducer Grouping.** After analyzing an approach that maximizes parallelism, we now move to the other end of the spectrum and design an algorithm that minimizes communication costs. In contrast to the Vanilla approach, where the complexity resides on the Map phase and the Reduce phase behaves as if implementing an ordinary GROUP BY clause, we propose an In-Reducer Grouping (IRG) approach, where all the logic of grouping is performed in the Reduce phase.

In-Reducer Grouping makes use of the possibility to define a *partitioner* in Hadoop [10, 20]. The mapper selects the columns of interest (in our example, all columns are needed, so the map function is simply the identity function). The keys are the finest granularity  $d_n$  (*day* in our example) but data is partitioned only by the columns of the coarsest granularity  $d_1$ . In this way, we can make sure that 1) each reducer receives enough data to compute the aggregation function even for the coarsest granularity  $d_1$ ; 2) the intermediate keys are sorted [10, 20], so for every grouping key of any granularity  $d_i$ , the reducer will process consecutively



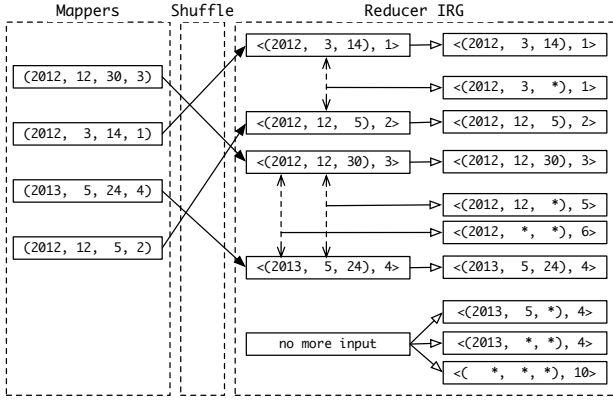


Figure 2: Example for the IRG approach.

all records pertaining to the given grouping key.

Figure 2 shows an example of the IRG approach. The mapper is the identity function, producing  $(year, month, day)$  as the keys and  $payload$  as the value. The coarsest granularity  $d_1$  is overall, and  $N_1 = 1$ : hence, all  $\langle key, value \rangle$  pairs are sent to a single reducer. The reducer groups all values of the same key, and processes the list of values associated to that key, thus computing the sum of all values as the total payload  $t$ . The grouping logic in the reducer also takes care of sending  $t$  to  $n$  grouping keys constructed from the reducer input key. For example, with reference to Figure 2, the input pair  $\langle (2012, 3, 14), 1 \rangle$  implies that value  $t = 1$  is sent to grouping keys  $(2012, 3, 14)$ ,  $(2012, 3, *)$ ,  $(2012, *, *)$  and  $(*, *, *)$ . The aggregators in these grouping keys accumulate all  $t$  values they receive. When there is no more  $t$  value for a grouping key (in our example, when  $year$  or  $month$  change, as shown by the dashed lines in Figure 2), the aggregator outputs the final aggregated value.

The key observation we exploit in the IRG approach is that a secondary, lexicographic sorting, is fundamental to minimize state in the reducers. For instance, at  $month$  granularity, when the reducer starts processing pair  $\langle (2013, 5, 24), 4 \rangle$ , then we are guaranteed that all grouping keys of month smaller than  $(2013, 5)$  (e.g.  $(2012, 12)$ ) have already been processed and should be output without further delay. This way reducers need not keep track of aggregators for previous grouping keys: reducers only use  $n$  aggregators, one for each granularity.

To summarize, the IRG approach extensively relies on the idea of an *on-line algorithm*: it makes a single pass over its input, maintaining only the necessary state to accumulate aggregates (both algebraic and holistic) at different granularities, and produces outputs as the reduce function iterates over the input.

**Parallelism and Communication Cost.** Since mappers output one tuple per input record, the replication rate of the IRG algorithm meets the lower bound of 1, as showed in Equation 2. On the other hand, this approach has limited parallelism, since it uses no more reducers than the number  $N_1$  of grouping keys at granularity  $d_1$ . In particular, when an overall aggregation is required, IRG can only work on a *single reducer*. As a result, IRG is likely to perform less work and require less resources than the Vanilla approach described previously, but it cannot benefit from paralleliza-

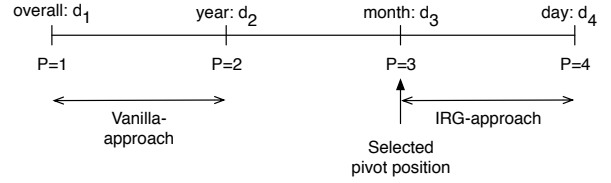


Figure 3: Pivot position example.

tion in the reduce phase.

**Impact of Combiners.** Since the IRG algorithm minimizes communication cost, combiners only perform well if pre-aggregation at the finest granularity  $d_n$  is beneficial – i.e., if the number of rows  $L$  in the data set is definitely larger than the number of grouping keys at the finest granularity,  $N_n$ . As such, the performance of the IRG approach suffers the least from the absence of combiners, e.g. when aggregation functions are not algebraic.

If the aggregate function is algebraic, however, the IRG algorithm is designed to re-use results from finer granularities in order to build the aggregation function *hierarchically*: in our running example, the aggregate of the total payload processed in a month can be obtained by summing the payload processed in the days of that month, and the aggregate for a year can likewise be computed by adding up the total payload for each month. Such an approach saves and reuses computation in a way that is not possible to obtain with the Vanilla approach.

**Hybrid approach: Vanilla + IRG.** We have shown that Vanilla and IRG are two “extreme” approaches: the first one maximizes parallelism at the expense of communication cost, the second one instead minimizes communication cost but does not provide good parallelism guarantees.

Neither approach is likely to be an optimal choice for such a tradeoff: in a realistic system, we are likely to have way less reducers than number of output tuples to generate (therefore making the extreme parallelism guarantees produced by Vanilla excessive); however, in particular when an overall aggregate is needed, it is reasonable to require an implementation that does not have the bottleneck of a single reducer.

In order to benefit at once from an acceptable level of parallelism and lower communication overheads, we propose an *hybrid* algorithm that fixes a *pivot* granularity  $P$ : all aggregate functions on granularities between  $d_P$  and  $d_n$  are computed using the IRG algorithm, while aggregates for granularities above  $d_P$  are obtained using the Vanilla approach. A choice of  $P = 1$  is equivalent to the IRG algorithm, while  $P = n$  corresponds to the Vanilla approach.

Let us consider again our running example, and fix the pivot position at  $P = 3$ , as shown in Figure 3. This choice implies that aggregates for the overall and  $year$  granularities  $d_1, d_2$  are computed using the Vanilla approach, while aggregates for the other granularities  $d_3, d_4$  ( $month$  and  $day$ ) are obtained using the IRG algorithm. For example, for the  $(2012, 3, 14, 1)$  tuple, the hybrid approach produces *three* output records at the mapper:

$(2012, 3, 14, 1)$  (day granularity)  
 $(2012, *, *, 1)$  (year granularity)  
 $( *, *, *, 1)$  (overall granularity)

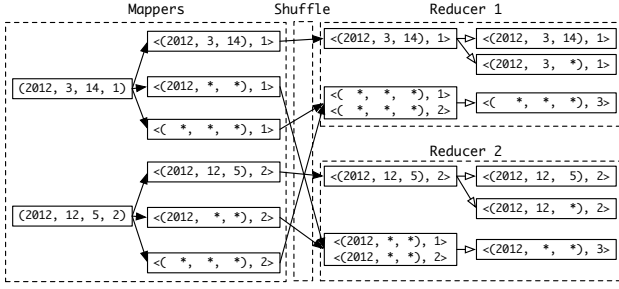


Figure 4: Example for the Hybrid Vanilla + IRG approach.

In this case the map output key space is partitioned by the month granularity, so that there is 1) one reducer per each month in the input dataset, that computes aggregates for granularities up to the month level, and 2) multiple reducers that compute aggregates for the *overall* and *year* granularities. Figure 4 illustrates an example with two reducers.

Some remarks are in order. Assuming a uniform distribution of the input dataset, the load on reducers of type 1) is expected to be evenly shared, as an input partition corresponds to an individual month and not the whole dataset. The load on reducers of type 2) might seem still prohibitive; however, we note that when combiners are in place they are going to vastly reduce the amount of data sent to the reducers responsible of the overall and *year* aggregate computation. For our example, the reducers of type 2) receive few input records, because the overall and *year* aggregates can be largely computed in the map phase. Furthermore, we remark that the efficiency of combiners in reducing input data to reducers (and communication costs) is very high for coarse granularities, and decreases towards finer granularities: this is why the IRG algorithm applies the Vanilla approach from the pivot position, up to coarse granularities. *Parallelism and Communication Cost.* The performance of the hybrid algorithm depends on the choice of  $P$ : the replication rate (before combiners) is  $P$ . The number of reducer that this approach can use is the total of 1)  $N_P$  grouping keys that are handled with the IRG algorithm, and 2)  $\sum_{i=0}^{P-1} N_i$  grouping keys that are handled with the Vanilla approach. Ideally, an *a priori* knowledge of the input data can be used to guide the choice of the pivot position. For example, if the data in our running example is known to span over tens of years and we know we only have ten reducer slots available (*i.e.*, at most ten reducer tasks can run concurrently), a choice of partitioning by year ( $P = 2$ ) would be reasonable. Conversely, if the dataset only spans a few years and hundreds of reducer slots are available, then it would be better to be more conservative and choose  $P = 3$  or  $P = 4$  to obtain better parallelism at the expense of a higher communication cost.

*Impact of Combiners.* The hybrid approach heavily relies on combiners. Indeed, when combiners are not available, all input data will be sent to the one reducer in charge of the overall granularity; in this case, it is then generally better to choose  $P = 1$  and revert to the IRG algorithm. However, when the combiners are available, the benefit for the hybrid approach is considerable, as discussed above.

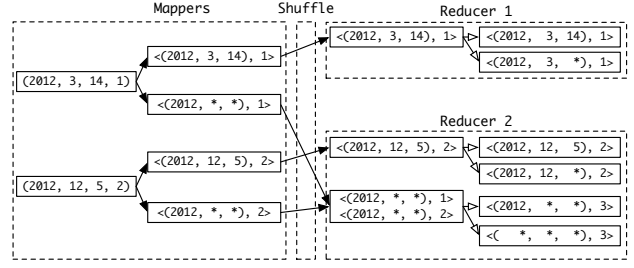


Figure 5: Example for the Hybrid IRG + IRG approach.

### 4.3 Alternative hybrid algorithms

We now extend the hybrid approach we introduced previously, and propose two alternatives: a single job involving two parallel IRG instances, and a chained job involving a first IRG computation and a final IRG aggregation.

**Hybrid approach: IRG + IRG.** In the previous section, we have shown that it is possible to design an algorithm aiming at striking a good balance between parallelism and replication rate, using a single parameter, *i.e.* the pivot position. In the baseline hybrid approach, parallelism is an increasing function of the replication rate, so that better parallelism is counterbalanced by higher communication costs in the shuffle phase.

Here, we propose an alternative approach that results in a constant replication rate of 2: the “trick” is to replace the Vanilla part of the baseline hybrid algorithm with a second IRG approach. Using the same running example as before, for the tuple (2012, 3, 14, 1), and selecting the pivot position  $P = 3$ , the *two* map output records are:

(2012, 3, 14, 1) (day granularity)  
(2012, \*, \*, 1) (year granularity)

Figure 5 illustrates a running example. In this case, the map output key space is partitioned by the *month* granularity, such that there is one reducer per month that uses the IRG algorithm to compute aggregates; in addition, there is *one reducer* receiving all tuples having ALL values taking care of the *year* and overall granularities, using again the IRG approach. As before, the role of combiners is crucial: the amount of (year, \*, \*, payload) tuples that are sent to the single reducer taking care of *year* and *overall* aggregates is likely to be very small, because opportunities to compute partial aggregates in the map phase are higher for coarser granularities.

*Parallelism and Communication Cost.* This algorithm has a constant replication rate of 2. As we show in Section 5, the choice of the pivot position  $P$  is here much less decisive than for the baseline hybrid approach: this can be explained by the fact that moving the pivot to finer granularities does not increase communication costs, as long as the load on the reducer taking care of the aggregates for coarse granularities remains low.

*Impact of Combiners.* Similarly to the baseline hybrid approach, this algorithm relies heavily on combiners; if combiners are not available, then, a simple IRG approach would be preferable.

**Chained IRG.** It is possible to further decrease the replication rate and hence the communication costs of computing ROLLUP aggregates by adopting a multi-round approach composed of two chained MapReduce jobs. In this case, the first job pre-aggregates results up to the pivot position  $P$  using the IRG algorithm; the second job uses partial aggregates from the first job to produce – on a single reducer – the final aggregate result, again using IRG. We note here that a similar observation, albeit for computing matrix multiplication, is also discussed in detail in [4].

*Parallelism and Communication Cost.* The parallelism of the first MapReduce job is determined by the amount  $N_P$  of grouping keys at the pivot position; the second MapReduce job, has a single reducer. However, the input size of the second job is likely to be orders of magnitude smaller than the first one, so that the runtime of the reduce phase of the second job – unless the pivot position puts too much effort on the second job – is generally negligible. The fact that the second reducer operates on a very small amount of input, results in a replication rate very close to 1.

The main drawback of the chained approach is due to job scheduling strategies: if jobs are scheduled in a system with idle resources, as we show in Section 5, the chained IRG algorithm results in the smallest runtime. However, in a loaded system, the second (and in general very small) MapReduce job could be scheduled later, resulting in artificially large delays between job submission and its execution.

*Impact of Combiners.* This approach does not rely heavily on combiners *per se*. However, it requires the aggregation function to be algebraic in order to make it possible for the second MapReduce job to re-use partial results.

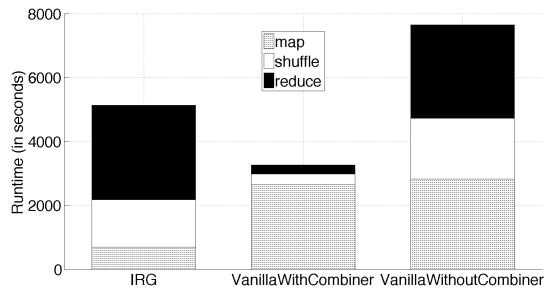
## 5. EXPERIMENTAL EVALUATION

We now proceed with an experimental approach to study the performance of the algorithms we discussed in this work. We use two main metrics: *runtime* – *i.e.* job execution time – and *total amount of work*, *i.e.* the sum of individual task execution times. Runtime is relevant on idle systems, in which job scheduling does not interfere with execution times; total amount of work is instead an important metric to study in heavily loaded systems where spare resources could be assigned to other pending jobs.

### 5.1 Experimental Setup

Our experimental evaluation is done on a Hadoop cluster of 20 slave machines (8GB RAM and a 4-core CPU) with 2 map and 1 reduce slot each. The HDFS block size is set to 128MB. All results shown in the following are the average of 5 runs: the standard deviation is smaller than 2.5%, hence – for the sake of readability – we omit error bars from our figures.

We compare the five approaches described in Section 4: baseline algorithms (Vanilla, IRG, Hybrid Vanilla + IRG) and alternative hybrid approaches (Hybrid IRG + IRG Chained IRG). We evaluate a single ROLLUP aggregation job over (*overall, year, month, day, hour, minute, second*) that uses the SUM aggregate function which, being algebraic, can benefit from combiners. Our input dataset is a synthetic log-trace representing historical traffic measurements taken by an Internet Service Provider (ISP): each record in our log has 1) a time-stamp expressed in (*year, month, day, hour, minute, second*); and 2) a number representing the payload (e.g. number of bytes sent or received



**Figure 6: Impact of combiners on runtime for the Vanilla approach.**

over the ISP network). The time-stamp is generated uniformly at random within a variable number of years (where not otherwise specified, the default is 40 years). The payload is a uniformly random positive integer. Overall, our dataset comprises 1.5 billion binary tuples of size 32 bytes each, packed in a SequenceFile [20].

### 5.2 Results

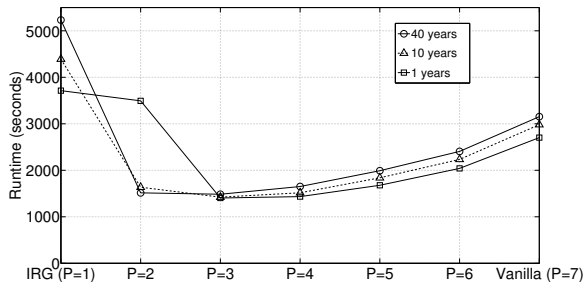
This section presents a range of results we obtained in our experiments. Before delving into a comparative study of all the approaches outlined above, we first focus on studying the impact of combiners on the performance of the Vanilla approach. Then, we move to a detailed analysis of runtime and amount of work for baseline algorithms (Vanilla, IRG, and Hybrid), and we conclude with an overview to outline merits and drawbacks of alternative hybrid approaches.

**The role of combiners.** Figure 6 illustrates a break-down of the runtime for computing the ROLLUP aggregate on our dataset, showing the time a job spend in the various phases of a MapReduce computation. Clearly, combiners play an important role for the Vanilla approach: they are beneficial in the shuffle and reduce phases. When combiners cannot be used (e.g. because the aggregation function is not algebraic), the IRG algorithm outperforms the Vanilla approach. With combiners enabled, the IRG algorithm is slower (larger runtimes) than the Vanilla approach: this can be explained by the lack of parallelism that characterizes IRG, wherein a single reducer is used as opposed to 20 reducers for the Vanilla algorithm. Note that, in the following experiments, combiners are systematically enabled. Finally, Figure 6 confirms that the IRG approach moves algorithmic complexity from the map phase to the reduce phase.

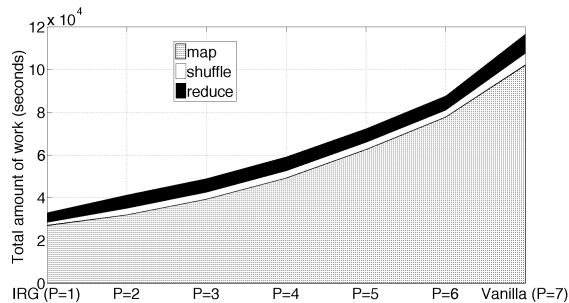
**Baseline algorithms.** In Figure 7(a) we compare the runtime of Vanilla, IRG, and the hybrid Vanilla + IRG approach. In our experiments we study the impact of the pivot position  $P$ , in lights of the “nature” of the input dataset: we synthetically generate data such that they span 1, 10 and 40 years worth of traffic logs.<sup>4</sup>

Clearly, IRG (which corresponds to  $P = 1$ ) is the slowest approach in terms of runtime. Indeed, using a single reducer incurs in prohibitive I/O overheads: the amount of data shuffled into a single reducer is too large to fit into mem-

<sup>4</sup>Note that the size – in terms of number of tuples – of the input data is kept constant, irrespectively of the number of represented years.



(a) Runtime



(b) Amount of work

Figure 7: Comparison of baseline approaches.

ory, therefore spilling and merging operations at the reducer proceed at disk speeds. Although no redundant computations are carried out in IRG, I/O costs outweigh the savings in computations.

A hybrid approach ( $2 \leq P \leq 6$ ) outperforms both IRG and Vanilla algorithms, with runtime as little as half that of the Vanilla approach. Communication costs make the runtime grow slowly as the pivot position moves towards finer granularities, suggesting that in doubt, it is better to position the pivot to the right (increased communication costs) rather than to the left (lack of parallelism). In our case, where a maximum of 20 reduce tasks can be scheduled at any time, our results indicate that  $P$  should be chosen such that  $N_P$  is larger than the number of available reducers. As expected, experiments with data from a single year indicate that the pivot position should be placed further to the right: the hybrid approach with  $P = 2$  essentially performs as badly as the single-reducer IRG.

Now, we present our results under a different perspective: we focus on the *total amount of work* executed by a ROLLUP aggregate implemented according to our baseline algorithms. We define the total amount of work for a given job as the sum of the runtime of each of its (map and reduce) tasks. Figure 7(b) indicates that the IRG approach consumes the least amount of work. By design, IRG is built to avoid redundant work: it has minimal replication rate, and the single reducer can produce ROLLUP aggregates with a single pass over its input.

As a general remark, that applies to all baseline algorithms, we note that the total amount of work is largely determined by the map phase of our jobs. The trend is tangible as  $P$  moves toward finer granularities: despite communication costs (the shuffle phase, which accounts for the replication rate) do not increase much with higher values of  $P$  thanks to the key role of combiners, map tasks still need to materialize data on disk before it can be combined and shuffled, thus contributing to a large extent to higher amounts of work.

**Alternative Hybrid Approaches.** We now give a compact representation of our experimental results for variants of the Hybrid approach we introduce in this work. Figure 8 offers a comparison, in terms of job runtime, of the Hybrid Vanilla + IRG approach to the Hybrid IRG + IRG and the Chained IRG algorithms. For the sake of readability, we omit from the figure experiments corresponding to  $P = 1$

and  $P = 7$ .

Figure 8 shows that the job runtime of the Hybrid Vanilla + IRG algorithm is sensitive to the choice of the pivot position  $P$ . Despite the use of combiners, the Vanilla “component” of the hybrid algorithm largely determines the job runtime, as discussed above. The IRG + IRG hybrid algorithm obtains lower job runtime and is less sensitive to the pivot position, albeit  $3 \leq P \leq 5$  constitutes an ideal range in which to place the pivot. The best performance in terms of runtime is achieved by the Chained IRG approach: in this case, the amount of data shuffled through the network (aggregated over each individual job of the chain) is smaller than what can be achieved by a single MapReduce job. We further observe that placing  $P$  towards finer granularities contributes to small job runtime: once an appropriate level of parallelism can be achieved in the first job of the chain, the computation cost of the second job in the chain is negligible, and the total amount of work (not shown here due to space limitations) is almost constant and extremely close to the one for IRG.

We can now summarize our findings as follows:

- All the approaches that we examined greatly benefit from the, fortunately common, property that aggregation functions are algebraic and therefore enable combiners and re-using partial results. If this is not the case, approaches based on the IRG algorithm are preferable.
- If total amount of work is the metric to optimize, IRG is the best solution because it minimizes redundant work. If low latency is also required, hybrid approaches offer a good trade-off, provided that the pivot position  $P$  is chosen appropriately.
- Our alternative hybrid approaches are the best performing solutions; both are very resilient to bad choices of the  $P$  pivot position, which can therefore be chosen with a very rough a-priori knowledge of the input dataset. Chained IRG provides the best results due to its minimal communication costs. However, chained jobs may suffer from bad scheduling decisions in a heavily loaded cluster, as the second job in the chain may “starve” due to large jobs being scheduled first. The literature on MapReduce scheduling offers solutions to this problem [18].

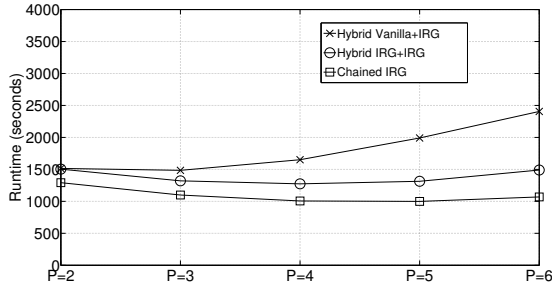


Figure 8: Comparison between alternative hybrid approaches.

## 6. CONCLUSION & FUTURE WORK

In this paper we have studied the problem of the efficient computation of ROLLUP aggregates in MapReduce. We proposed a modeling approach to untangle the available design space to address this problem, by focusing on the trade-off that exists between the achievable parallelism and communication costs that characterize the MapReduce programming model. This was helpful in identifying the limitations of current ROLLUP implementations, that only cover a small portion of the design space as they concentrate solely on parallelism. We presented an algorithm to meet the lower bounds of the communication costs we derived in our model, and showed that minimum replication can be achieved at the expenses of parallelism. In addition we presented several variants of ROLLUP implementations that share a common trait: a single parameter (the pivot) allows tuning the parallelism vs. communication trade-off for finding a reasonable “sweet spot”.

Our work was enriched by an experimental evaluation of several ROLLUP implementations. The experimental approach revealed the importance of optimizations currently available in systems such as Hadoop, which could not be taken into account with a modeling approach alone. Our experiments showed, in addition to the performance of each ROLLUP variant in terms of runtime, that the efficiency of the new algorithms we designed in this work was superior to what is available in the current state of the art.

Our plan is to extend our experimental evaluation to consider skewed datasets: we believe that our hybrid algorithms exhibit the distinguishing feature that the pivot position can be used not only to gauge parallelism and replication, but also to mitigate the possible uneven computational load distribution when data is not uniform. We also consider a data-dependent pivot, which is an even more refined pivot than our current schema-dependent one. Furthermore, we plan to extend our work by designing an automatic mechanism to select an appropriate pivot position, depending on the nature of the data to process.

## Acknowledgments

The authors would like to thank Antonio Barbuzzi for his valuable comments. This work has been partially supported by the EU project BigFoot (FP7-ICT-317858).

## 7. REFERENCES

- [1] <http://hadoop.apache.org>.
- [2] <http://hive.apache.org>.
- [3] <http://pig.apache.org>.
- [4] F. N. Afrati et al. Upper and lower bounds on the cost of a map-reduce computation. In *VLDB*, 2013.
- [5] F. N. Afrati and J. D. Ullman. Optimizing Multiway Joins in a Map-Reduce Environment. *IEEE Transactions on Knowledge and Data Engineering*, 2011.
- [6] S. Agarwal et al. On the Computation of Multidimensional Aggregates. In *VLDB*, 1996.
- [7] S. Bellamkonda et al. Adaptive and Big Data Scale Parallel Execution in Oracle. In *VLDB*, 2013.
- [8] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *ACM SIGMOD*, 1999.
- [9] S. Blanas et al. A comparison of join algorithms for log processing in MapReduce. In *ACM SIGMOD*, 2010.
- [10] J. Dean and S. Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. In *ACM OSDI*, 2004.
- [11] M. Fang et al. Computing Iceberg Queries Efficiently. In *VLDB*, 1998.
- [12] J. Gray et al. Data Cube : A Relational Aggregation Operator Generalizing Data Cube : A Relational Aggregation Operator Generalizing Group-By , Cross-Tab , and Sub-Totals. *Data Mining and Knowledge Discovery*, 1997.
- [13] J. Hah, J. Pei, and G. Dong. Efficient Computation of Iceberg Cubes with Complex Measures. In *ACM SIGMOD*, 2001.
- [14] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD*, 1996.
- [15] P. Jayachandran. Implementing RollupDimensions UDF and adding ROLLUP clause in CUBE operator. PIG-2765 JIRA.
- [16] A. Nandi et al. Distributed cube materialization on holistic measures. In *IEEE ICDE*, 2011.
- [17] A. Okcan and M. Riedewald. Processing Theta-Joins using MapReduce. In *ACM SIGMOD*, 2011.
- [18] M. Pastorelli et al. HFSP: Size-based scheduling for hadoop. In *IEEE BigData*, 2013.
- [19] K. A. Ross and D. Srivastava. Fast computation of sparse datacubes. In *VLDB*, 1997.
- [20] T. White. *Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale*. O’Reilly, 2012.

# Determining the $k$ in $k$ -means with MapReduce

Thibault Debatty

Royal Military Academy, Brussels, Belgium  
thibault.debatty@rma.ac.be

Wim Mees

Royal Military Academy, Brussels, Belgium  
wim.mees@rma.ac.be

Pietro Michiardi

EURECOM, Campus SophiaTech, France  
pietro.michiardi@eurecom.fr

Olivier Thonnard

Symantec Research Labs, Sophia Antipolis, France  
olivier\_thonnard@symantec.com

## Abstract

In this paper we propose a MapReduce implementation of G-means, a variant of  $k$ -means that is able to automatically determine  $k$ , the number of clusters. We show that our implementation scales to very large datasets and very large values of  $k$ , as the computation cost is proportional to  $nk$ . Other techniques that run a clustering algorithm with different values of  $k$  and choose the value of  $k$  that provides the “best” results have a computation cost that is proportional to  $nk^2$ .

We run experiments that confirm that the processing time is proportional to  $k$ . These experiments also show that, because G-means adds new centers progressively, if and where they are needed, it reduces the probability to fall into a local minimum, and finally finds better centers than classical  $k$ -means processing.

## 1. INTRODUCTION

Discovering groups of similar objects in a dataset, also known as clustering, is one of the most fundamental techniques of data analysis [12]. Clustering algorithms are used in many fields including machine learning, pattern recognition, image analysis, information retrieval, market segmentation and bioinformatics.

A lot of different algorithms exist, mainly depending on their definition of a cluster. Density based algorithms, like DBSCAN [8] and OPTICS [2] for example, define a cluster as a high density region in the feature space. Other algorithms assume that the data is generated from a mixture of statistical distributions. Finally, centroid models, like  $k$ -means, represent each cluster by a single center point. This algorithm thus implicitly assumes that the points in each cluster are spherically distributed around the center [9].

The most known algorithm for computing  $k$ -means clustering is Lloyd’s algorithm [13], also known as “the  $k$ -means algorithm”. Although, it was published more than 30 year ago, it is still widely used today as it is at the same time simple and effective [12]. However, it also has a number of

drawbacks:

1. It may converge to a local minimum, producing counterintuitive or even inconsistent results.
2. It is not really efficient, and may converge very slowly.
3. It prefers clusters of approximately similar size, as it always assigns an object to the nearest center. This often leads to incorrect borders between clusters.
4. Finally, like a lot of other clustering algorithms, it requires the number of clusters –  $k$  – to be specified in advance, which is considered as one of the most difficult problems to solve in data clustering [12].

In this paper we tackle this last drawback. We present and analyze the performance of a MapReduce implementation of G-means[9], an efficient algorithm to determine  $k$ . We also compare our algorithm to a common MapReduce implementation of  $k$ -means.

More specifically, we first show that a MapReduce implementation of G-means requires some modifications of the original algorithm to reduce I/O operations, as these are very costly in MapReduce, and to reduce the number of chained MR jobs. We also show that an efficient implementation that maximizes processing parallelism requires a hybrid design that takes into account the number of nodes running the algorithm and the quantity of heap memory available.

We then study the performance aspects of the proposed algorithm implementation by modeling the communication and computational cost. We show that our algorithm is able to determine  $k$  and find clusters with a computation cost proportional to  $nk$ . Other techniques that run a clustering algorithm with different values of  $k$  and choose the value of  $k$  that provides the “best” results have a computation cost that is proportional to  $nk^2$ .

Finally, we evaluate both solutions experimentally. Our results confirm that the proposed MR implementation of G-means has linear complexity with respect to  $k$ . The algorithm also takes full advantage of additional computing nodes, which makes it scalable to very large datasets. Moreover, our experiments show that our implementation clearly outperforms the classical iterative  $k$ -means solution as it reduces the probability to fall into a local minimum and provides better clustering results.

The rest of the paper is organized as follows : In section 2 we present G-means[9] and other existing methods to determine  $k$ , as well as other optimizations of  $k$ -means. In

section 3 we present and justify our MapReduce implementation of G-means. In section 4 we estimate and compare the computation and communication costs of the MapReduce implementations of G-means and  $k$ -means. In section 5 we present our experimental results, and finally we present our conclusions.

## 2. RELATED WORK

When clustering a dataset, the right number of clusters to use –  $k$  – is often a parameter of the algorithm.

Even when analyzing data visually, the correct choice of  $k$  is often ambiguous. It largely depends on the shape and scale of the distribution of points in the data set and on the desired clustering resolution of the user.

In addition, arbitrarily increasing  $k$  will always reduce the amount of error in the resulting clustering, to the extreme case of zero error if each data point is considered its own cluster.

If an appropriate value of  $k$  is not apparent from prior knowledge of the properties of the data set, it must be chosen somehow. There are several methods for making this decision. Lots of them rely on cluster evaluation metrics. They run a clustering algorithm with different values of  $k$ , and choose the value of  $k$  that provides the “best” results according to some criterion.

For example, Dunn’s index (DI) [7] can be used to determine the number of clusters. The  $k$  for which the DI is the highest can be chosen as the number of clusters.

The elbow method [20] is another possible criterion. It chooses a number of clusters so that adding another cluster doesn’t give much better modeling of the data. Therefore, it computes the percentage of variance explained (the ratio of the between-group variance to the total variance, also known as an F-test) for different values of  $k$ . In the graph of the percentage of variance explained by the clusters against the number of clusters, the first clusters will add much information (explain a lot of variance), but at some point the marginal gain will drop, giving an angle in the graph. The number of clusters is chosen at this point, hence the “elbow criterion”. As it is a visual method, this “elbow” cannot always be unambiguously identified.

The average silhouette of the data [18] is another useful criterion for assessing the natural number of clusters. The silhouette of a point is a measure of how close it is to other points within its cluster and how loosely it is matched to points of the neighboring cluster, i.e. the cluster whose average distance from the point is lowest. A silhouette close to 1 implies the point is in an appropriate cluster, while a silhouette close to -1 implies the point is in the wrong cluster. If there are too many or too few clusters, as it may occur when a wrong value of  $k$  is used with  $k$ -means algorithm, some of the clusters will typically display much narrower silhouettes than the rest. Thus silhouette plots and averages may also be used to determine the natural number of clusters within a dataset.

Sugar and James [19] used information theory to propose a new index of cluster quality, called the “Jump method”. The method is based on the notion of “distortion”, which is a measure of within-cluster dispersion. For each possible value of  $k$ , the method calculates the “jump” of distortion compared with previous value of  $k$ . The Estimated number of clusters is the value of  $k$  with the largest jump.

Tibshirani and al. [21] proposed another method based

on dispersion, called the “Gap statistic” for estimating the number of clusters in a data set. The idea is to compare the change in within-cluster dispersion to that expected under an appropriate null distribution as reference. The number of clusters is then the value for which the observed dispersion falls the farthest below the expected dispersion obtained under a null distribution.

Finally, two other studies presented iterative techniques to determine the number of clusters when performing  $k$ -means clustering, which do not require to run  $k$ -means for every possible value of  $k$ : X-means [17] and G-means [9].

X-means iteratively uses  $k$ -means to optimize the position of centers and increases the number of clusters if needed to optimize the Bayesian Information Criterion (BIC) or the Akaike Information Criterion (AIC) measure. The main advantage of the algorithm is the efficiency of the test used to select the most promising centers for refinement. This leads to a fast algorithm that outputs both the number of clusters and their position. Experiments showed this technique revealed the true number of clusters in the underlying distribution, and that it was much faster than repeatedly using  $k$ -means for different values of  $k$ .

G-means is also an iterative algorithm but it uses Anderson-Darling test to verify whether a subset of data follows a Gaussian distribution. G-means runs  $k$ -means with increasing values of  $k$  in a hierarchical fashion until the test accepts the hypothesis that the points assigned to each center follow a Gaussian distribution. Experimental results showed that the algorithm seems to outperform X-means.

The G-means algorithm starts with a small number of clusters, and increases the number of centers. At each iteration, the algorithm runs  $k$ -means to refine the current centers. The clusters whose data appears not to come from a Gaussian distribution are then split.

For each cluster  $X$  (being a subset of data) of center  $c$ , the algorithm works as follows:

1. Find two new centers  $c_1$  and  $c_2$ .
2. Run  $k$ -means to refine  $c_1$  and  $c_2$ .
3. Let  $v = c_1 - c_2$  be the vector that connects the two centers. This is the direction that  $k$ -means believes is important for clustering.
4. Let  $X'$  be the projection of  $X$  on  $v$ .  $X'$  is a one-dimensional representation of the data projected on  $v$ .
5. Normalize  $X'$  so that it has zero mean and variance equal to 1.
6. Use Anderson-Darling to test  $X'$ :
  - If  $X'$  follows a normal distribution, keep the original center, and discard  $c_1$  and  $c_2$ .
  - Otherwise, split the cluster in two, use  $c_1$  and  $c_2$  as new centers and run the algorithm on each sub-cluster.

The main advantage of this algorithm is that it simplifies the test for Gaussian fit by projecting the data to one dimension where the test is simple to apply. Moreover it only creates new centers where needed, improving clustering quality.

In this paper, we present a MapReduce implementation of the G-means algorithm. Some key challenges to be addressed are the various design choices for parallelizing the algorithm, as these may have a significant impact on final results quality, but also on communication and computational cost.

While the choice of  $k$  is a critical question, many other optimizations have been proposed in the literature to improve or speed up  $k$ -means processing.

A first optimization consists in selecting better initial centers, which allows the algorithm to converge quicker, reduces the probability to fall into a local minimum and reduces the number trials needed. In  $k$ -means++ [3], the starting centers are chosen randomly, but with a probability proportional the distance to the nearest already chosen center. Bahmani [4] also proposed a MapReduce version of  $k$ -means++ initialization algorithm. Another common possibility is to use canopy clustering [15] to compute the initial centers. Algorithms also exist to avoid local minimums, for example by swapping points between clusters [11].

Other optimizations deal with nearest neighbor (NN) search. In  $k$ -means, a NN search is required to decide to which cluster a point belongs. It is thus one of the basic operations of  $k$ -means processing, but also of a lot of other clustering algorithms. One type of efficient NN search algorithm uses tree-based structures, like the mrkd-tree algorithm proposed by Pelleg et al. [16]. The algorithm uses a multi-resolution  $k$ -d tree to represent groups of points and efficiently identify the nearest cluster centers for those points. Vrahatis et al. [22] proposed a version that uses a windowing technique based on range trees. A range tree on a set of points in  $d$ -dimensions is a recursively defined multi-level binary search tree. Each level of the range tree is a binary search tree on one of the  $d$ -dimensions, which allows fast range searches. Another category of algorithms uses random projection, like Locality Sensitive Hash used by Buhler [5].

Other algorithms improve the clustering efficiency by first summarizing a large data set, and then applying the clustering algorithm. Different approaches exist:

- Replace a small tight group of objects (but not the whole cluster) by a single object [6] or by a coresets [10];
- Pre-process data to reduce dimensionality, dropping unnecessary features (dimensions) [1];
- Partition data into overlapping subsets [15] for high dimensional data.

While all these different optimizations of  $k$ -means are definitively valuable, it is outside the scope of this paper to implement and evaluate all of them. However, some of these optimizations could be easily integrated in the MapReduce implementation proposed in this paper, and we are considering them as part of our future work.

### 3. MAPREDUCE IMPLEMENTATION OF G-MEANS

Our implementation of G-means for MapReduce is presented in Algorithm 1.

The first step, `PickInitialCenters`, is a classical step of any  $k$ -means algorithm. The main difference with respect

---

#### Algorithm 1 MapReduce G-means pseudo-code

---

```

PICKINITIALCENTERS
while Not CLUSTERINGCOMPLETED do
    KMEANS
    KMEANSANDFINDNEWCENTERS
    TESTCLUSTERS
end while

```

---

to classical  $k$ -means implementations is that it picks pairs of centers ( $c_1$  and  $c_2$ ). We use a serial implementation, that picks initial centers at random, but other distributed or more efficient algorithms can be found in the literature and can perfectly be used instead.

The algorithm then enters a `while` loop that will continue as long as there are clusters that must be split. The first operation of the loop is a classical MapReduce implementation of  $k$ -means with combiners<sup>1</sup>, to refine to position of current centers.

The last iteration of  $k$ -means is implemented in a separate MapReduce job called `KMeansAndFindNewCenters` in Algorithm 2. It will also, for each cluster, pick the two new centers ( $c_1$  and  $c_2$ ) that will be used at next iteration. This job is specific to our implementation and is further explained below.

Finally, the clusters are tested using the MapReduce job referred to as `TestClusters` in Algorithm 1. For each point, the job searches the cluster it belongs to (using the centers from previous iteration), then projects it on the vector formed by the two corresponding centers (of current iteration). Finally, for each cluster it tests if the projections form a normal distribution. This job, also specific to the proposed implementation, is explained in more details here below.

As can be noticed, our MapReduce implementation of G-means differs from the sequential version in three main aspects.

First, the original G-means algorithm works locally, by analyzing each cluster separately. It thus requires that each point is “linked” in some way to the cluster it belongs to at each iteration of the algorithm. Implementing this in MapReduce would require a write operation at each iteration, to save this information in the distributed file system.

This membership information can of course be used to reduce computations at some steps of the algorithm:

- When running  $k$ -means, for each point, the algorithm does not have to compute the distance to each center, but only to  $c_1$  and  $c_2$ , the 2 children centers of the cluster the point currently belongs to;
- When testing the clusters, the cluster to which a point belongs is directly identified, and the algorithm does not have to compute the distance from this point to each cluster.

However, binding the points to their cluster would require a write operation at each iteration, and could at best spare  $O(2nk)$  distance computations. Given the very high cost of I/O operations in MapReduce, we do not recommend using this solution. Moreover, as mentioned above, other techniques already exist to optimize nearest neighbor search that can perfectly be added to our implementation.

<sup>1</sup>A combiner is a well-known pre-aggregation optimization available in MapReduce.



Next, in the original G-means algorithm, new centers are picked at the beginning of each iteration. Implementing this directly in MapReduce would require an additional MapReduce job. To minimize the number of jobs executed at each iteration and the number of dataset reads, we merge this operation with the last iteration of  $k$ -means. Thus, the `KMeansAndFindNewCenters` operation will perform classical  $k$ -means and at the same time find 2 new centers ( $c_1$  and  $c_2$ ) for each cluster, which will be used at next iteration of G-means.

Finally, while the sequential algorithm analyzes clusters individually, and thus adds new centers sequentially, the MapReduce version analyzes all clusters in parallel and will thus try to double the number of centers at each iteration. As a result, it may eventually overestimate the value of  $k$ . Future versions of the algorithm will thus add a post-processing step to merge close centers.

One of the subtleties of the MapReduce version of G-means, as proposed in Algorithm 1, is that each iteration has to deal with centers from previous, current and next iteration:

- `KMeans` refines the centers of current iteration;
- `KMeansAndFindNewCenters` picks centers that will be used at next iteration;
- `TestClusters` assigns each point to its cluster (a center from previous iteration), then projects it on the vector joining the 2 corresponding centers of current iteration.

### 3.1 KMeans and Find New Centers

`KMeansAndFindNewCenters` is a MapReduce job with combiners that performs two operations at the same time:

1. Run  $k$ -means to refine current centers;
2. For each current center, pick two new centers ( $c_1$  and  $c_2$ ) that will possibly be used at next iteration.

In our implementation, the new centers are chosen randomly. More sophisticated algorithms can be used to select the new points, but they may require an additional MapReduce job.

---

#### Algorithm 2 KMeansAndFindNewCenters Mapper

---

Input: *point* (text)  
Output:  
*centerid* (long)  $\Rightarrow$  coordinates (float[]), 1 (int)  
*centerid* + *OFFSET* (long)  $\Rightarrow$  coordinates (float[]), 1 (int)

```

procedure MAP(key, point)
  Find nearest center
  Emit(centerid, point)
  Emit(centerid + OFFSET, point)
end procedure

```

---

The Map step of the job is presented in Algorithm 2. The coordinates of each point are emitted twice. This doubles the quantity of data to be shuffled and transmitted over the network. However, this effect is largely mitigated by the use of a combiner. The efficiency of the combiner is of course very dependent of the dataset. There are recent execution

engines (such as SPARK<sup>2</sup>) that allow to specify "partition-preserving" operations. Preserving partitions would help the combiners to perform more efficiently at next iteration. It is however outside the scope of this paper to consider such optimizations.

To make the distinction between coordinates that correspond to new centers to be used at next iteration of the algorithm and current centers that we want to refine with  $k$ -means, we use an arbitrary high offset value. More precisely, as the type of `center id` is a Java Long, we use an offset value equal to half the largest possible value of a Java Long. The value of `OFFSET` is thus  $2^{62}$  (approximately  $4E18$ ). This also limits our algorithm to datasets with at most  $2^{62}$  centers.

We could also use a text prefix, but although simpler to interpret, this choice would hurt performance due to the requirement of an additional parsing phase. Moreover, during the shuffle phase, sorting text keys requires more processing than simple integer values.

The combiner and reducer test the value of the key. If it is larger than the predefined offset, they keep only 2 new centers per cluster. Otherwise they perform classical  $k$ -means reduction and compute the new position of each cluster center.

### 3.2 Test Clusters

The `TestClusters` procedure is the last MapReduce job of our distributed G-means implementation (Algorithm 1). The mapper projects the points of a cluster on the line joining the two centers ( $c_1$  and  $c_2$ ) and the reducer then tests if these values follow a normal distribution.

---

#### Algorithm 3 TestClusters Mapper

---

Input: *point* (text)  
Output: *vectorid* (int)  $\Rightarrow$  *projection* (double)

```

procedure SETUP
  Build vectors from center pairs
  Read centers from previous iteration
end procedure

procedure MAP(key, point)
  Find nearest center
  Find corresponding vector
  Compute projection of point on vector
  Emit(vectorid, projection)
end procedure

```

---



---

#### Algorithm 4 TestClusters Reducer

---

Input: *vectorid* (int)  $\Rightarrow$  *projection* (double) >

```

procedure REDUCE(vectorid, projections)
  Read projections to build a vector
  Normalize vector (mean 0, stddev 1)
  ADTEST(vector)
  if normal then
    Mark cluster as found
  end if
end procedure

```

---

<sup>2</sup><http://www.spark-project.org/>

At the first steps of G-means, when  $k$  is low, this algorithm performs poorly as the parallelism of the reduce phase is bounded by  $k$ .

To achieve higher parallelism, the algorithm adopts another strategy when  $k$  is low, called **TestFewClusters** (Algorithm 5). The test for normality is directly performed by the mapper, thus on subsets of data. This of course only delivers correct results if the number of samples for each subset is sufficient, which we can suppose is verified for low values of  $k$ . Anderson-Darling is a powerful statistical test, which has proved being reliable even with small samples (as a rule of thumb, a minimum size of 8 is considered to be sufficient). In our implementation we use a threshold of 20, to stay on the safe side. The number of reduce tasks is still equal to  $k$  (which is low), but as their task is only to combine the decisions taken by mappers, this will not limit the performance of the algorithm.

---

**Algorithm 5** TestFewClusters Mapper

---

Input: *point* (text)  
Output: *vectorid* (int)  $\Rightarrow A^{*2}$  (double)

**procedure** SETUP  
  Build vectors from center pairs  
  Read centers from previous iteration  
**end procedure**

**procedure** MAP(*key*, *point*)  
  Find nearest *center*  
  Find corresponding *vector*  
  Compute *projection* of *point* on *vector*  
  Add *projection* to *list vectorid*  
**end procedure**

**procedure** CLOSE  
  **for** Each *list do*  
    Read projections to build a *vector*  
    Normalize *vector* (mean 0 , stddev 1)  
    Compute  $A^{*2} = \text{adtest}(\text{vector})$   
    Emit(*vectorid*  $\Rightarrow A^{*2}$ )  
  **end for**  
**end procedure**

---

Moreover, **TestFewClusters** limits the size of the vector of projections to a level that fits into RAM memory: If we assume that the value of a point in each dimension is stored as a string of approximately 15 characters (the number of significant decimal digits of IEEE 754 double-precision floating-point format), and each character is encoded using 1 Byte, the number of points in a dataset is  $O(\frac{S}{15D})$ , where  $S$  is the size of the dataset (in Bytes) and  $D$  is the number of dimensions.

For each point, the algorithm will compute a projection, encoded as a double (8 Bytes). The total memory space needed to store all projections is thus  $O(8\frac{S}{10D})$  and thus  $O(\frac{S}{D})$  Bytes, which can be very large. In the worst case scenario, if all points of the dataset belong to the same cluster, as a result of the **TestClusters** procedure, the amount of memory required by a single combiner will be prohibitive.

When **TestFewClusters** is used, the quantity of memory required by each mapper to store the projections will be  $O(\frac{Ss}{D})$ , where  $Ss$  is the size of a single split (64MB on a

default Hadoop installation), which is now completely reasonable.

Choosing when to switch from one strategy to the other is, as often, a matter of compromise.

If the algorithm switches too late (i.e., when  $k$  is large), the algorithm will keep using the **TestFewClusters** strategy, even for a large number of clusters. As the test for normality is performed by the mapper, there is a risk that the number of points in some clusters is smaller than the threshold. The mapper is then not able to compute a decision.

If the algorithm switches too early (i.e., when  $k$  is small), the test is performed by the reducers even for a small number of clusters. There is a risk that the number of projections received by a single reducer becomes too large and exhausts the heap: in the worst case, the maximum amount of memory required by a single reducer will be  $O(\frac{S}{D})$  Bytes (if the complete dataset belongs to a single cluster), and in the best case it will be  $O(\frac{S}{kD})$  Bytes (if all  $k$  clusters have the same number of points).

In our MapReduce implementation of G-means, at each iteration the algorithm counts the number of points that belong to each cluster. By doing so, the algorithm can estimate the maximum amount of heap memory that will be required as the number of points belonging to the biggest cluster multiplied by the average quantity of heap memory required per point (that we determined experimentally).

When an algorithm uses almost all heap memory available, the Java Virtual Machine (JVM) has to regularly trigger the garbage collector to make room for new objects and variables, which seriously degrades performance. To avoid this, we use a maximum heap usage coefficient.

The algorithm will thus first use the **TestFewClusters** strategy, and switch to the other strategy only when the following two conditions are met: the number of clusters to test is larger than the total reduce capacity, and the estimated maximum amount of required heap memory is less than 66% of the heap memory of the JVM.

As illustration, Figure 1 shows the centers found by successive iterations of our final MapReduce G-means algorithm for a subset of data, consisting of 10 clusters in  $\mathbb{R}^2$ . At each iteration the algorithm splits clusters in 2, except clusters that pass the test, and optimizes centers position using  $k$ -means. The algorithm finally finds 14 centers, as shown in Figure 4.

## 4. COST MODELIZATION

We now estimate the cost of MapReduce G-means clustering. More precisely, we estimate the number of dataset reads, the number of computations and the quantity of data that is shuffled.

Each iteration of G-means consists of three steps: **KMeans**, **KMeansAndFindNewCenters**, and **TestClusters**.

Each iteration of **KMeans** requires one dataset read<sup>3</sup>,  $O(kn)$  distance computations, and shuffles  $O(n)$  coordinates in worst case (if no combiner is used). As the new centers are placed in an efficient way, where they are really needed, we found experimentally that only two  $k$ -means iterations are sufficient.

<sup>3</sup>Depending on the underlying execution engine, it may be possible to avoid subsequent dataset reads. This is the case for example with SPARK, where you can cache the dataset in memory and make sure to preserve the data partitioning.

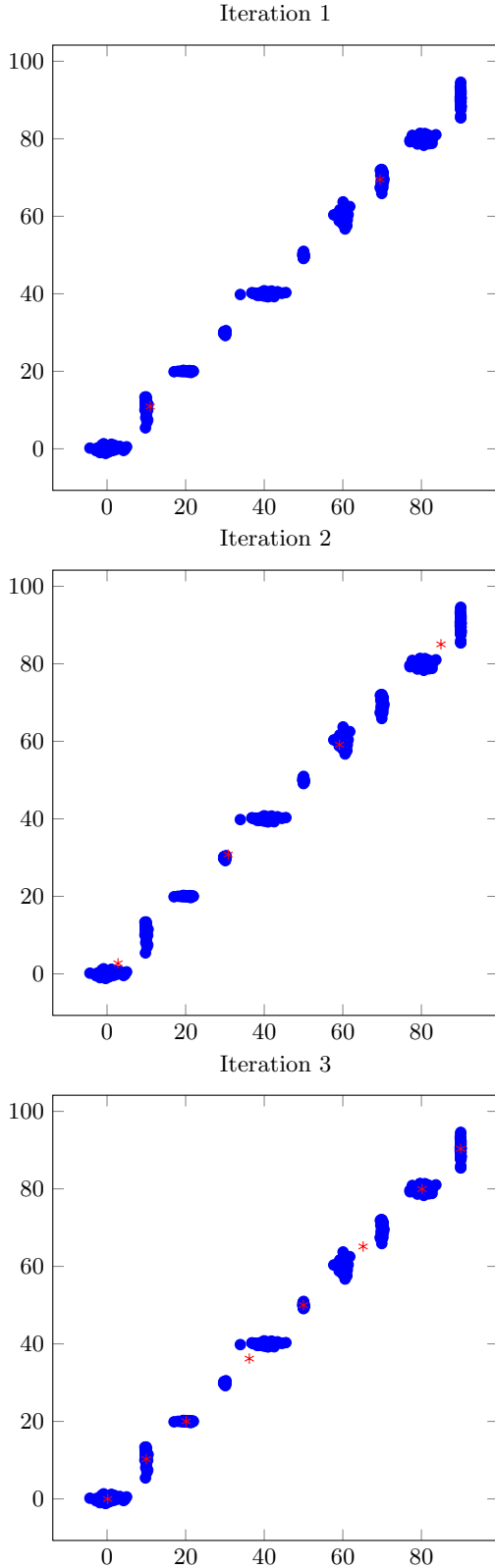


Figure 1: Evolution of centers positioned by G-means in a dataset containing 10 clusters in  $\mathbb{R}^2$

**KMeansAndFindNewCenters** consists of a single  $k$ -means iteration, but the mapper will emit each point a second time to pick two new centers for each current center. It also requires one dataset read,  $O(kn)$  distance computations and, without combiner, shuffles  $O(2n)$  coordinates.

**TestClusters** requires one dataset read. It computes  $O(kn)$  distances,  $O(n)$  projections and performs  $O(k)$  anderson-darling tests. For large values of  $k$  (thus for a large number of clusters), this will be dominated by distances computation and anderson-darling tests. The step also shuffles  $O(n)$  projections.

If the algorithm starts with a single cluster at iteration 0, at iteration  $i$  it is updating  $2^{i+1} = 2k$  centers to test  $2^i = k$  possible clusters. At iteration  $i$ , the total number of clusters that have been tested is

$$1 + 2 + 4 + \dots + 2^i = \sum_{j=0}^i 2^j = 2^{i+1} - 1$$

The number of iterations required to test values of  $k$  between 1 and  $k_{real}$  is theoretically

$$n = \log_2 k_{real}$$

In practice a few additional iterations are required because MapReduce G-means tends to overestimate the number of clusters, and because some clusters are discovered before others.

The  $\sum k$  for all iterations of G-means is:

$$\sum_{j=0}^n k = \sum_{j=0}^n 2^j = 2^{n+1} - 1 \simeq O(2^{\log_2 k_{real} + 1} - 1) = O(2k_{real})$$

In total, G-means algorithm requires  $O(4 \log_2 k_{real})$  dataset reads, computation of  $O(4n \sum k) = O(8nk_{real})$  distances and  $O(\sum k) = 2k_{real}$  anderson-darling tests.

The algorithm is thus able to find  $k$  with a number of computations that remains proportional to  $k_{real}$ ! The price to pay is an iterative processing, that requires  $O(\log_2 k_{real})$  iterations, and thus  $O(\log_2 k_{real})$  dataset reads.

At the other side, the classical way to find  $k$  is to use a MapReduce implementation of  $k$ -means, to let it run for different values of  $k$ , and to use one of the criteria described above to find the bet value of  $k$ . However, this is not efficient.

To compare MapReduce versions of  $k$ -means and G-means in a fair way, we used another implementation, multi- $k$ -means, that computes the centers for all possible values of  $k$  at each iteration. The mapper step is presented by algorithm 6. The combiner and reducer are classical.

---

**Algorithm 6** Multi- $k$ -means Mapper

---

Input: *point* (text)

Output:  $k\_centerid$  (text)  $\Rightarrow$  *coordinates* (float[]), 1 (int)

**procedure** MAP(*key*, *point*)

**for**  $k = k\_min; k \leq k\_max; k += k\_step$  **do**

    Find nearest *center*

    Emit( $k\_centerid \Rightarrow point$ )

**end for**

**end procedure**

---

The main drawback is of course that number of distances

computed and the quantity of data that is shuffled and transmitted over the network at each iteration of  $k$ -means are much bigger. But the quantity of data to shuffle is largely reduced by using the combiner. So this drawback is largely outbalanced by the advantage that all possible values of  $k$  can be tested in a single round, thus vastly reducing the number of iterations and dataset reads!

To test all values of  $k$  between 1 and  $k_{max}$ , the total number of centers computed by multi- $k$ -means is:

$$\sum_{j=1}^{k_{max}} j = \frac{k(k+1)}{2} \simeq O(k^2)$$

At each iteration, multi- $k$ -means requires 1 dataset read,  $O(nk_{max}^2)$  distance computations and shuffles  $O(nk_{max})$  coordinates if no combiner is used.

Clearly, from a theoretical point of view G-means has a huge advantage over multi- $k$ -means, as the number of computations remains proportional with  $k_{real}$  instead of  $k_{max}^2$ . It does, however need  $O(\log_2 k_{real})$  iterations, and thus  $O(\log_2 k_{real})$  dataset reads.

For example, for a dataset containing 100 clusters, G-means theoretically requires 7 iterations, and thus  $O(800n)$  distance computations,  $O(200)$  anderson-darling tests and 28 dataset reads. At the other side, for such a small value, multi- $k$ -means already requires  $O(10000n)$  distance computations at each iteration!

Moreover, G-means stops processing when  $k$  is found, while multi- $k$ -means has to process all possible values of  $k$  before taking a decision. As G-means adds new centers progressively, where they are required, it reduces the probability to get stuck in a local minimum, while this can be the case for multi- $k$ -means if initial centers are poorly chosen. A production version of multi- $k$ -means thus requires multiple runs with different starting points, or an additional job to select initial centers, for example using canopy clustering[15], or an algorithm that avoids local optima [11]. Finally, once the centers have been computed for different values of  $k$ , multi- $k$ -means requires at least one additional job to find the correct value of  $k$ .

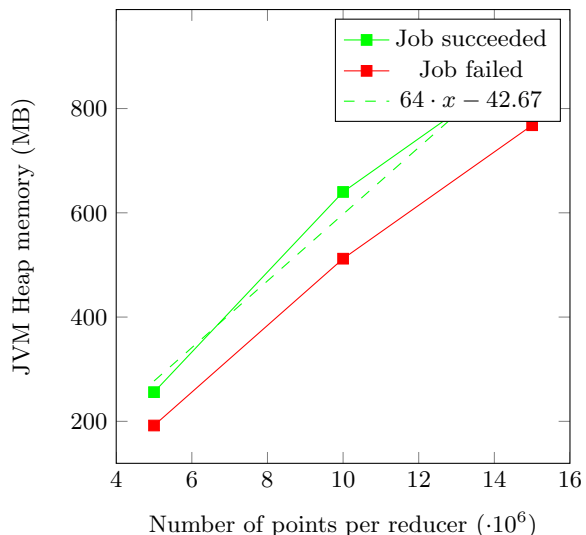
In any way, there is a risk that because of skewed data, some reducers will have a higher workload, thus reducing the global efficiency of the algorithm. Handling skewed data in MapReduce is a whole subject by itself and is left as future work.

## 5. EXPERIMENTAL RESULTS

To test the algorithms we propose in this paper, we use a Hadoop implementation and run tests on a cluster consisting of 4 nodes. Each node is equipped with 2 quad-core Xeon processors and 32GB of RAM.

As a first experiment, we want to estimate the quantity of heap memory required by the reducer of the `TestClusters` step of Algorithm 1. Therefore we run the algorithm with different datasets that consist of a variable number of points. During the first iteration of the algorithm, all points belong to a single cluster, and will thus be tested by a single reducer. We iteratively run the algorithm, and let the amount of heap memory vary. When the quantity of available heap memory becomes too small, the job crashes with an error ("Java heap space"). The results are shown on Figure 2.

Linear regression shows our reducer requires approximately 64 Bytes (8 doubles) per point. This value can cer-



**Figure 2: Estimation of the amount of heap memory required by the reducer of the step `TestClusters`**

tainly be further optimized, but it is not the goal of this paper to create a production level version of the algorithm. So for all other tests, the algorithm uses that value of 64 to estimate the quantity of heap memory required by the reducer of the `TestClusters` step, and to decide when to switch from one strategy to the other.

We now turn to the experiments performed in order to test our G-means algorithm on different synthetic datasets. We used five datasets of 10M points (in  $\mathbb{R}^{10}$ ) generated using a Gaussian distribution, and using a variable number of clusters ranging from 100 up to 1600. Table 1 shows for each dataset the real number of clusters, the number of clusters discovered by G-means, as well as the number of iterations and time required.

**Table 1: Results of G-means clustering**

	d100	d200	d400	d800	d1600
Clusters	100	200	400	800	1600
Discovered	134	305	626	1264	2455
Time (sec)	1286	1667	2291	4208	5593
Iterations	9	10	11	13	13

As expected, the algorithm overestimates the number of clusters. The proportion of discovered clusters to the real number of clusters seems to be quite constant (1.5). The algorithm thus requires a post-processing step to merge clusters, the development of which is left as future work.

The number of iterations is also slightly greater than the theoretical value ( $1 + \log_2 k$ ). As some centers are discovered before the last iteration, the algorithm will not create new centers for them. It may thus require 1 or more additional iterations to discover all centers.

Finally, as expected, the execution time scales linearly with  $k$ .

We then compare G-means with a hadoop implementation of multi- $k$ -means. For each dataset, multi- $k$ -means computes the position of centers for all values of  $k$  between one

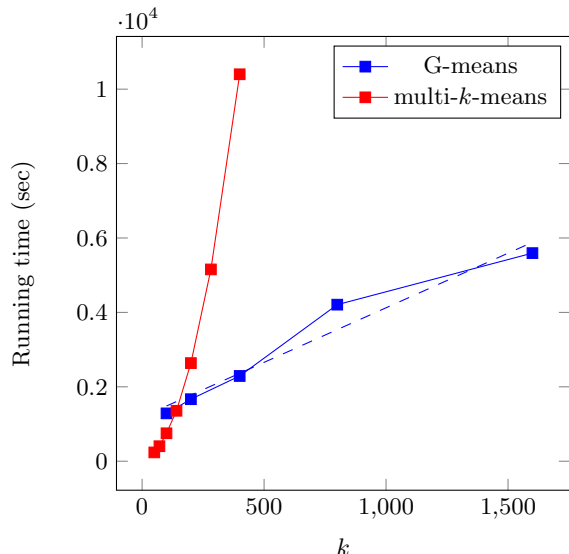
and the real number of clusters in the dataset. Table 2 shows the average computing time for a single iteration.

**Table 2: Average time of a single iteration of multi- $k$ -means**

	d50	d100	d141	d200	d400
Clusters	50	100	141	200	400
Time (sec)	237	751	1356	2637	10252

The execution time of both G-means and multi- $k$ -means are graphed in Figure 3.

We observe now that the execution time of multi- $k$ -means rises exponentially. Moreover, for a single iteration of multi- $k$ -means, and for a value of  $k$  as low as 100, G-means already outperforms multi- $k$ -means.



**Figure 3: Running time of G-means and multi- $k$ -means**

We also want to evaluate the consistency of the clustering results provided by both algorithms. The final goal of  $k$ -means clustering is to partition the  $n$  data points into  $k$  sets  $S = S_1, S_2, \dots, S_k$  so as to minimize the within-cluster sum of squares (WCSS):

$$\arg \min_S \sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

where  $\boldsymbol{\mu}_i$  is the mean of points in  $S_i$ . Therefore we use WCSS as a measure of the quality of clustering.

For different datasets, Table 3 shows the real number of clusters in the dataset, the number of clusters discovered by G-means, the average distance between points and their centers discovered by G-means, and the average distance between points and centers when using multi- $k$ -means for the same value of  $k$ . A smaller value means the position of the centers is better. For multi- $k$ -means, we let the algorithm run 10 iterations, which is enough to find a stable solution.

As can be seen in Table 3, G-means consistently outperforms multi- $k$ -means, by approximately 10%. This is

**Table 3: Real number of clusters in each dataset, number of clusters discovered by G-means, and average distance between points and centers found by G-means and multi- $k$ -means**

	d100	d200	d400
$k_{real}$	100	200	400
$k_{found}$	150	279	639
G-means	3.34	3.33	3.23
multi- $k$ -means	3.71	3.60	3.39

mainly due to the fact G-means progressively adds new centers, if and where they are needed. This reduces the probability to fall into a local minimum.

This effect is illustrated on Figure 4. Both plots show a small dataset consisting of 10 clusters. The upper plot shows the 14 centers found by G-means. This is more than the real number of clusters, but the clusters are correctly detected. The lower plot shows the centers found by multi- $k$ -means after 10 iterations, for  $k = 10$ . Two centers are located in the penultimate cluster, around (80, 80). Although the multi- $k$ -means searches the position of the correct number of centers, it falls into a local minimum and does not detect the correct clusters, which results in a larger average distance between point and center.

Finally, to test the scalability of the algorithm, we generate a dataset consisting of 100M points (in  $\mathbb{R}^{10}$ ) distributed in 1000 clusters using a Gaussian distribution. We then run our MR G-means algorithm on 4, 8 and 12 nodes. All tests completed after 13 iterations of G-means. The respective running times are shown in Table 4 and on Figure 5.

**Table 4: Running time of MR G-means to cluster a dataset of 100M points**

	T4	T8	T12
Nodes	4	8	12
Time (min)	798	447	323

We can observe that the running time decreases roughly linearly with the number of nodes, which shows that our algorithm and Hadoop can take advantage of additional nodes and thus scale to very large datasets.

## 6. CONCLUSIONS AND FUTURE WORK

Despite its known drawbacks and alternative techniques,  $k$ -means [14] is still a largely used clustering algorithm. It also has a lot of variants and optimizations. One of these variants, G-means, is able to automatically determine  $k$ , the number of clusters. In this paper we proposed a MapReduce implementation of G-means that is able to estimate  $k$  with a computation cost that is proportional to  $k$ .

We ran experiments that confirm that the processing time scales linearly with  $k$ . These experiments also show that, because G-means adds new centers progressively, if and where they are needed, it reduces the probability to fall into a local minimum, and eventually finds better centers than classical  $k$ -means processing.

The algorithm still has some caveats, as it tends to constantly overestimate the number of clusters, but it definitely

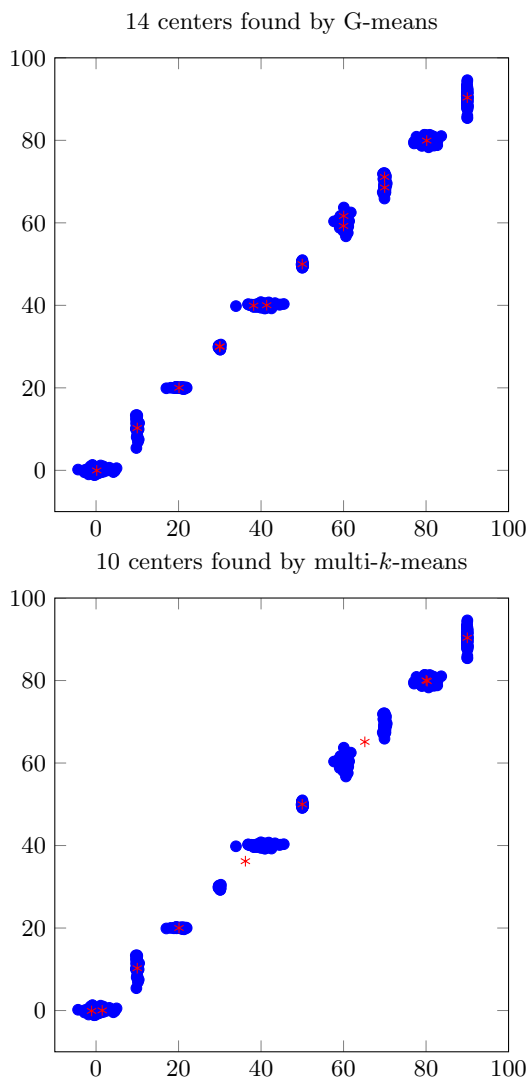


Figure 4: Results of MR G-means and multi- $k$ -means

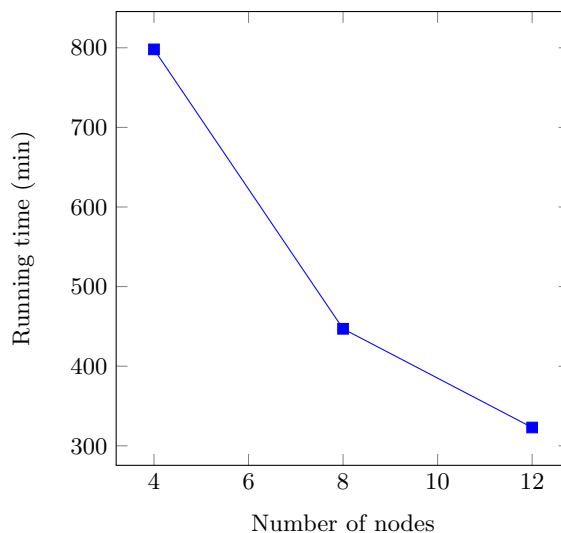


Figure 5: Running time of MR G-means to cluster a dataset of 100M points

deserves interest when it comes to clustering large datasets consisting of an unknown number of clusters.

As future work, we plan to explore ways to extend our MapReduce implementation of G-means by leveraging more advanced batch execution engine (e.g. SPARK) which can provide advanced configuration options at run-time in order to save unnecessary disk I/O operations via in-memory caching and partition-preserving operations. We also plan to run additional experiments on a larger cluster to evaluate further the performance and scalability of the algorithm.

## Acknowledgement

This work has been partially supported by the EU project BigFoot (FP7-ICT-317858).

## 7. REFERENCES

- [1] Charu C. Aggarwal, Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. *SIGMOD Rec.*, 28(2):61–72, June 1999.
- [2] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28(2):49–60, June 1999.
- [3] David Arthur and S Vassilvitskii. k-means++: The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.
- [4] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow.*, 5(7):622–633, March 2012.
- [5] Jeremy Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17(5):419–428, 2001.
- [6] Rudi Cilibrasi and Paul M. B. Vitányi. Clustering by compression. *IEEE Transactions on Information Theory*, 51:1523–1545, 2005.

- [7] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.
- [8] Martin Ester, Hans-Peter Kriegel, Jörg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.
- [9] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Neural Information Processing Systems*. MIT Press, 2003.
- [10] S. Har-Peled and S. Mazumdar. Coresets for  $k$ -means and  $k$ -median clustering and their applications. pages 291–300, 2004.
- [11] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A  $k$ -means clustering algorithm. *Applied Statistics*, 28(1), 1979.
- [12] Anil K Jain. Data Clustering : 50 Years Beyond K-Means. *Pattern Recognition Letters*, 2009.
- [13] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.
- [14] J. MacQueen. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1965/66, 1, 281-297 (1967)., 1967.
- [15] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.
- [16] Dan Pelleg and Andrew Moore. Accelerating exact  $k$ -means algorithms with geometric reasoning. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999.
- [17] Dan Pelleg and AW Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. *ICML*, 2000.
- [18] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(0):53 – 65, 1987.
- [19] Catherine A Sugar and Gareth M James. Finding the number of clusters in a dataset. *Journal of the American Statistical Association*, 98(463):750–763, 2003.
- [20] RobertL. Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.
- [21] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.
- [22] M.N. Vrahatis, B. Boutsinas, P. Alevizos, and G. Pavlides. The New  $k$ -Windows Algorithm for Improving the  $k$ -Means Clustering Algorithm. *Journal of Complexity*, 18(1):375–391, March 2002.

# Tagged Dataflow: a Formal Model for Iterative Map-Reduce\*

Angelos Charalambidis  
University of Athens  
a.charalambidis@di.uoa.gr

Nikolaos Papaspyrou  
National Technical University  
of Athens  
nickie@softlab.ntua.gr

Panos Rondogiannis  
University of Athens  
prondo@di.uoa.gr

## ABSTRACT

In this paper, we consider the recent iterative extensions of the Map-Reduce framework and we argue that they would greatly benefit from the research work that was conducted in the area of dataflow computing more than thirty years ago. In particular, we suggest that the *tagged-dataflow* model of computation can be used as the formal framework behind existing and future iterative generalizations of Map-Reduce. Moreover, we present various applications in which the tagged model gives elegant solutions with increased parallelism. The tagged-dataflow approach for iterative Map-Reduce creates a number of interesting research challenges which deserve further investigation, such as the requirement for a more sophisticated fault tolerance model.

## 1. INTRODUCTION

The introduction of Map-Reduce [11] has been an important step towards the efficient processing of massive data. The success of the Map-Reduce framework is mainly due to its simplicity, its declarative nature, its ability to run on commodity clusters and its effective handling of task failures that occur during execution. Despite its huge success, Map-Reduce has often been criticized for a number of different reasons. For example, it has often been argued that not all problems can be effectively (and naturally) solved using map and reduce tasks. Moreover, it has often been stressed that the framework suffers from a lack of support for *iteration*, which is, without doubt, a cornerstone of most interesting forms of computation.

The realization of the above problems has led to the introduction of various extensions of the framework (see for

---

\*This research was supported by the project “Handling Uncertainty in Data Intensive Applications”, co-financed by the European Union (European Social Fund) and Greek national funds, through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Program: THALES, Investing in knowledge society through the European Social Fund.

example [16, 7, 8]) and to the development of new systems that try to overcome the shortcomings (such as for example [13, 19, 14, 27, 20, 9, 21]). It appears that the creation of a framework that preserves the advantages of Map-Reduce and at the same time lifts its shortcomings, is not an easy task. In particular, it seems that the clear and efficient support of iteration is far from straightforward. Furthermore, there does not exist at present a unifying theoretical framework that could form the basis of the iterative extensions of Map-Reduce. Such a framework would allow the theory of such systems to be properly developed. In particular, it would allow for a proper specification and analysis of proposed algorithms, for the development of a corresponding complexity theory, for the proof of correctness of proposed algorithms, and so on.

In this paper we argue that such a formal framework was actually developed more than thirty years ago, in a similar but somewhat more restricted context. More specifically, during the 70s and 80s, the *dataflow model* of computation was developed [10, 12], many dataflow architectures were built and several dataflow programming languages were proposed and implemented. The aim of the dataflow paradigm was to exploit the inherent parallelism that exists in many applications. The dataflow architectures were specialized parallel computers and the dataflow programs were designed to eventually run on such machines. Of course, the Internet was not fully developed at that time and the idea of parallelism over clusters of workstations did not exist. Moreover, the distributed processing of massive data was far less important at that time than it is today. However, the main ideas behind the dataflow programs of the past and the distributed Map-Reduce applications of the present, share many common characteristics. In particular, the effective handling of iteration was also an important problem in the dataflow era and was handled by the introduction of the so-called paradigm of *tagged-dataflow* [25, 5, 6].

The main contribution of the present paper is therefore to demonstrate how the ideas from the dataflow research area can be applied to the new area of iterative Map-Reduce. In particular, we demonstrate how the tagging schemes of dataflow computers can be used in order to achieve a more asynchronous form of iteration for the systems of today. The rest of the paper is organized as follows. Section 2 summarizes several extensions of Map-Reduce and discusses the way iteration is handled in these extensions. Section 3 presents the basic ideas of the traditional model of tagged-dataflow. Section 4 extends the classical tagged-dataflow model so as to be applicable to the Map-Reduce setting and demonstrates



how the new model can be used in order to specify various applications on massive data that appear to require iteration in their implementation. Finally, Section 5 describes possible directions for future work.

## 2. ITERATIVE MAP-REDUCE

Despite its simplicity and usefulness, Map-Reduce [11] has certain shortcomings that restrict its wider applicability. First, the framework allows only two types of tasks, namely the map and reduce tasks; obviously, it is not always possible (or natural) to model any problem using these two types of tasks. The flow of the data is also rather rigid since the output of the map tasks is used as input only to the reduce tasks. Again, there exist problems whose distributed solution requires a more involved flow of data: in many applications the data must be processed iteratively before the final output is obtained. Finally, Map-Reduce imposes the so-called *blocking property*: the reducers start processing data when the mappers have completely finished their work. This property is crucial in order to deal with task-failures: when a task fails, we can restart it from the beginning. Therefore, the blocking property ensures that the data are not “garbled” and that the computation proceeds in clear, discrete steps. However, the blocking property obviously has its disadvantages. During the time that the map tasks are processing, the reduce tasks are idle waiting for the map computation to complete and this obviously reduces the amount of potential parallelism.

### 2.1 Iteration

A natural generalization of Map-Reduce is to allow the use of arbitrary tasks. The systems Dryad [16] and Hyracks [7] generalize Map-Reduce by allowing the use of a set of arbitrary operators connected as a directed acyclic graph.

Apart from generalizing the types of tasks, a significant extension of Map-Reduce is the support of iteration. Many common data analysis algorithms require some form of iteration in order to be appropriately implemented. For example, the PageRank algorithm is a recursive algorithm that is usually implemented as an iteration until a fixed point is reached. One can attempt to implement such algorithms in the Map-Reduce framework using ad hoc techniques, but this is neither a natural nor efficient approach. HaLoop [8] tackles this inadequacy of Map-Reduce by providing iteration-related constructs that allow iterative algorithms to be expressed more succinctly. A characteristic of HaLoop (as well as other similar systems) is that iteration is performed in a synchronized manner, i.e., the next iteration starts when every task of the previous iteration has completely finished its work. A shortcoming of this approach is that tasks that have completed their processing remain idle until the next iteration starts, and therefore the potential parallelism is not fully exploited.

There have also been proposals for a more asynchronous form of iteration. For example, Afrati *et al.* in [3] perform a theoretical investigation of the iterative execution of recursive Datalog queries in an extended Map-Reduce environment. In that work, the task graph is not necessarily acyclic and the execution of tasks need not be synchronized. In other words, in the framework of [3] the blocking property is lifted.

Recapitulating, it appears that in current and future extensions of Map-Reduce, iteration is a vital component. It

also appears that iteration can either be synchronous or asynchronous, depending on the objectives for which a particular system has been built.

## 2.2 Iterative Stream Processing

The problem of iteration becomes much more challenging when combined with the processing of stream data or stream queries. In many applications, the data to be processed are not always fully available at the time of execution. For example, in a social network, the edges and the vertices of the friends-of-friends graph can be added or removed dynamically. In such cases, the data are often most valuable as soon as they are produced (i.e., it is not possible for the data to be delayed and processed later as part of a batch). This state of affairs leads to the quest for iterative extensions of Map-Reduce that also take into account the temporal nature of the incoming data. A similar situation occurs when we would like to process a stream of different queries over the same data. If we would like to process the queries asynchronously (i.e., start processing a query before the previous one has completed execution), and if each query involves iterative computations, then care must be taken so as that the data produced during the processing of one query will not affect the processing of the other ones.

Certain extensions of Map-Reduce have been proposed that can handle situations such as the above. The system D-Streams [27] handles streaming input in a fault tolerant manner. The Naiad system realizes the concept of the differential dataflow [20] in which operators act upon “difference input traces” (i.e., the set of changes with respect to the previous input) in order to produce difference traces of output. The output then can be constructed by combining all the difference traces. The difference traces are indexed both by the version of the input and the iteration number. As a result the computation of an iterative incremental algorithm is drastically reduced since the framework can reuse computations done both in previous versions of the input and in previous iterations. Hadoop Online [9] is an extension of Map-Reduce that supports pipelined stream queries. In [21] multiple similar Map-Reduce jobs are combined in order to be executed as one. The key idea is to add tags in the data, in order to distinguish between different jobs.

The systems mentioned in this last subsection appear to be using a common technique in order to handle iteration in streaming data or queries: they employ some form of *tagging* in order to discriminate between the data that belong to different iteration levels and different versions of the input (or different queries). In the next sections we argue that this tagging mechanism is not just a coincidence, but instead a more general mechanism that can be used in order to implement arbitrary iteration in a generalized Map-Reduce framework.

## 3. TAGGED-DATAFLOW

The *dataflow model of computation* [10, 12] was developed more than thirty years ago, as an alternative to the classical “von-Neumann” computing model. The key motivation was the creation of architectures and programming languages that would exploit the massive parallelism that is inherent in many applications. A dataflow program is essentially a directed graph in which vertices correspond to processing elements and edges correspond to channels. The data that need to be processed start “flowing” inside the

channels; when they reach a node they are being processed and the data produced are fed to the output channels of the node. Since various parts of the dataflow graph can be working concurrently, the parallel nature of the model should be apparent. Moreover, this processing of data “while in motion” comes in sharp contrast with the traditional “von-Neumann” model in which data wait passively in memory until they are fetched by the central processing unit of the (sequential) computer in order to be processed.

The dataflow model was extensively studied during the 70s and 80s. Many theoretical results were obtained, many dataflow programming languages were developed and several dataflow machines were built. In the beginning of the 90s, the dataflow research area started to decline. The main reason was the fact that dataflow hardware never proved extremely successful and never achieved a performance that would justify the massive production and use of dataflow machines. However, the theory and the programming languages that were developed were quite sophisticated, leaving much hope for further developments in case the hardware problems were ever bypassed.

In the initial dataflow model (usually called *pipeline dataflow*), channels were assumed to be unbounded FIFO queues, i.e., the data were assumed to flow in a specific order inside the channels. However, it soon became apparent that a model that would not impose any particular temporal ordering of the data would be much more general and useful. This resulted in the so-called *tagged-dataflow* model [25, 5, 6]. The basic idea behind tagged-dataflow is that data can flow inside a network accompanied by *tags* (i.e., labels). The use of tags makes dataflow much more asynchronous since data need not be processed in any particular predetermined order. Moreover, as we are going to see, the tags can carry essential information that can be used in order to implement iterative or even recursive algorithms. A key notion in our foregoing discussion is that of a *dataflow graph*:

*Definition 1.* A *dataflow graph* (or *dataflow network*) is a directed graph  $G = (V, E)$ , where  $V$  is the set of *nodes* of the graph and  $E$  is the set of edges connecting elements of  $V$ . The set  $V$  is partitioned into disjoint subsets  $V_I$  (*input nodes*),  $V_O$  (*output nodes*) and  $V_P$  (*processing nodes*), subject to the following restrictions:

- Every input node has no incoming edges and has one outgoing edge towards a processing node.
- Every output node has no outgoing edges and has one incoming edge from a processing node.
- Every processing node has incoming edges (at least one) from input nodes and/or from other processing nodes and outgoing edges (at least one) to output nodes and/or other processing nodes.

Intuitively, input nodes provide the input data to the dataflow graph, processing nodes are performing the processing of data and output nodes are collecting the output data produced by the network.

The semantics of dataflow networks can be given with standard techniques of denotational semantics [23]. Our presentation below follows the exposition given in [26]. Intuitively, edges of our dataflow networks carry tuples of the form  $\langle t, d \rangle$  where  $d$  is an element of a data domain  $D$  and  $t$  is an element of a set of tags  $T$ . The set  $T$  may be quite

involved; in its simplest form it can be a set of natural numbers, or in more demanding cases it can be the set of lists of natural numbers, etc. Pairs of the form  $\langle t, d \rangle \in T \times D$  are usually referred in the dataflow literature as *tokens*. It is often assumed that for a given tag  $t$ , an edge can contain at most one token  $\langle t, d \rangle$ . In other words, it is a standard assumption that edges correspond to functions in  $T \rightarrow D$ . As we are going to see in the next section, this assumption needs to be extended when considering applications in the Map-Reduce framework.

Consider now a processing node of our dataflow graph. A standard assumption in dataflow computing is that processing nodes are functions that transform their inputs to outputs. There have been extensions of dataflow that support non-functional nodes, for example, non-deterministic ones [1]. However, such extensions will not be considered here. It should be noted that determinism is also a key assumption in Map-Reduce systems (see, for example, [11, page 109]). Therefore, based on the foregoing discussion, a processing node  $f$  of a dataflow network that has  $n \geq 1$  inputs and  $m \geq 1$  outputs is a function<sup>1</sup> in  $(T \rightarrow D)^n \rightarrow (T \rightarrow D)^m$ . The input and output nodes are in fact equivalent to channels, i.e., they are functions in  $T \rightarrow D$ .

In the rest of the paper we will refer to the class of dataflow networks presented above as *functional dataflow networks* (since channels are functions). We will present an extension of this model in the next section.

*Example 1.* We demonstrate these ideas with the well-known example of Hamming numbers (first posed as a programming problem by Edsger Dijkstra). Recall that Hamming numbers are all numbers of the form  $2^i \cdot 3^j \cdot 5^k$  where  $i, j, k$  are non-negative integers. The problem is to enumerate the Hamming numbers in numeric order and a dataflow solution of it is depicted in Figure 1. We explain the nodes that appear in the figure. First consider the nodes that contain constants (such as 1, 2, etc). One can assume that these nodes produce constant streams. For example, the node 5 is nothing more than the constant function in  $\mathbb{N} \rightarrow \mathbb{N}$  which assigns to every  $n \in \mathbb{N}$  the constant value 5. The *fbv* node (read as “followed-by”), is a function in  $(\mathbb{N} \rightarrow \mathbb{N}, \mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ . The operation of *fbv* can be intuitively described as follows: initially, it checks its first input until a token of the form  $\langle 0, d \rangle$  arrives, and as soon as it sees such a token, it delivers it to its output channel. From that point on, *fbv* never consults again its first input but it continuously checks its second input. Every time a token of the form  $\langle t, d \rangle$  arrives in its second input, the node puts in its output channel the token  $\langle t + 1, d \rangle$ . It can be easily checked that this operational behavior corresponds to the following functional definition of *fbv*:

$$fbv(X, Y)(t) = \begin{cases} X(0) & \text{if } t = 0 \\ Y(t - 1) & \text{if } t > 0 \end{cases}$$

The *merge* node is also a function in  $(\mathbb{N} \rightarrow \mathbb{N}, \mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$ . Given two inputs  $X, Y \in \mathbb{N} \rightarrow \mathbb{N}$  that are increasing functions, *merge* produces as output an increasing function that results from merging the two inputs. For the

<sup>1</sup>Actually, computability reasons dictate that the functions corresponding to the nodes of our networks have to be additionally *continuous* (see [23] for a standard introduction on this issue and [18] for a corresponding discussion regarding pipeline dataflow networks).

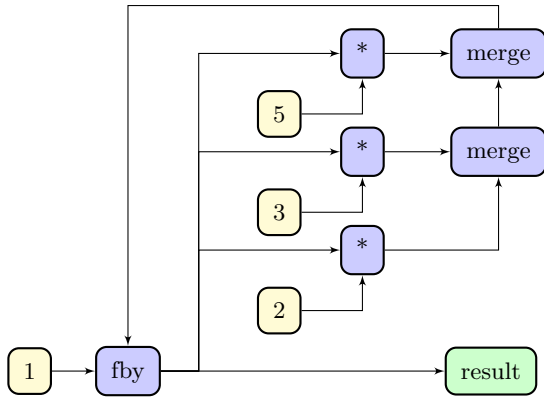


Figure 1: The dataflow representation of Hamming numbers.

formal (recursive) definition of *merge*, see [24]. The “\*” nodes simply multiply the data part of the token that arrives in their input, with the constant that exists in their other input.

One can easily verify that the network just described, produces the sequence of Hamming numbers.

## 4. TAGGED-DATAFLOW AND ITERATIVE MAP-REDUCE

In this section we argue that a mild extension of the tagged-dataflow model of computation can be used as the formal framework behind existing and future iterative generalizations of Map-Reduce. We present various applications in which the tagged model gives elegant solutions with increased asynchronous parallelism.

It is important to clarify what exactly is being claimed by the following examples. First, we argue that the *implementations* of iterative Map-Reduce systems should support tags and tag-manipulation operations in the same way that tagged-dataflow machines of the 80s supported such operations [15, 6]. The tags can be used to ensure the implementation of asynchronous iteration in an elegant and effective way. The main idea is precisely defined in the following excerpt from [15]:

*Each separate (loop) iteration reuses the same code but with different data. To avoid any confusion of operands from the different iterations, each data value is tagged with a unique identifier known as the iteration level that indicates its specific iteration. Data are transmitted along the arcs in tagged packets known as tokens.*

The second thing that is being claimed below is that the languages used to program iterative Map-Reduce applications should also support the declaration and manipulation of tags. In this way the programmer will be free to declare and use the types of tags that are essential in the specific application being developed. It is important to stress that dataflow languages of the 80s and 90s allowed the declaration of user-defined tags (also called *dimensions*). For example, the latest versions of Lucid [24] and its extension GLU [17] allowed many different dimensions.

### 4.1 An Extension of Tagged-Dataflow for Map-Reduce

The basic principles of tagged-dataflow described in the previous section have been used in the design of many functional dataflow programming languages. In particular, the interpretation of channels as functions and of processing nodes as (second-order) functions, clearly emphasizes the connections between dataflow networks and functional programming.

However, there exist applications in which it is not sufficient for channels to be just functions from  $T$  to  $D$ . For example, it is conceivable for the same tag to be used in two different tuples that “flow” inside a channel (i.e., to have  $\langle t, d_1 \rangle$  and  $\langle t, d_2 \rangle$  appear in the same channel). Moreover, it is possible in certain cases to have the same tuple  $\langle t, d \rangle$  appear more than once in a channel. In particular, as we are soon going to see, in the Map-Reduce framework both of these cases show-up quite naturally.

We are thus led to a generalization of the tagged-dataflow model in which channels are *multisets* over  $T \times D$  and processing nodes take multisets as inputs and produce multisets as outputs. More formally, given a nonempty set  $S$  let us denote by  $\mathcal{M}(S)$  the set of multisets (or *bags*) of elements of  $S$ . Then, in our extended tagged-dataflow model, a channel is an element of  $\mathcal{M}(T \times D)$  while a node  $f$  of a dataflow network that has  $n$  inputs and  $m$  outputs is a function in  $[\mathcal{M}(T \times D)]^n \rightarrow [\mathcal{M}(T \times D)]^m$ . In the following subsections, we examine three different applications where this extended tagged model is applicable and useful.

### 4.2 Streaming Queries

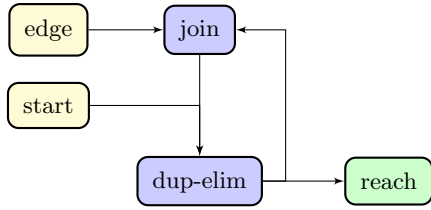
One of the most promising applications of iterative Map-Reduce is in the execution of Datalog queries [3, 4]. The key idea here is that the least fixed point of a Datalog program can be computed in a bottom-up way using a network of *join* and *dup-elim* processes (see [3] for details). For example, assume that we have an EDB relation *edge* and also the following IDB relation:

```
reach(Y) :- start(Y).
reach(Y) :- edge(X,Y), reach(X).
```

Assume also that we want to locate all the nodes that are reachable from an initial vertex *a*, i.e., we assume that we also have the fact *start(a)*. Using the techniques of [3], one can easily construct a simple network that calculates the set of nodes reachable from *a*.

Assume however that we want to have multiple queries, e.g., we want to locate all the vertices reachable from *a*, *b* and *c*. Moreover, for reasons of efficiency, we want these queries to be run in parallel as much as possible (which means for example that the query for *b* should not wait for the computation of the query for *a* to complete before starting to execute itself). The problem with running the queries in an overlapped manner is obvious: the output of the Map-Reduce network will be a set of *reach* facts, but with no indication of which fact corresponds to which query. So, if we want to do some further processing on the nodes reachable from vertex *a*, we have no way of knowing which exactly these nodes are.

In the tagged setting, the support of such streaming queries is quite straightforward. Each different *start* query is tagged with a different natural number. Then, every time one of the *reach* rules is used to produce some new fact, the tag that



**Figure 2:** The set of join and dup-elim nodes working with streaming queries. Queries flow from the start node tagged with a distinct number.

has been used in the body of the rule is inherited by the fact that is produced in the head of the rule. In this way, at the end of the computation a set of tuples  $\langle t, \text{reach}(\mathbf{u}) \rangle$  is collected, and one can discriminate the results of the different queries by examining the tag  $t$ .

It is not hard to see how the above example fits the tagged-model introduced in Subsection 4.1. The dataflow is depicted in Figure 2. The channels of our network are elements of  $\mathcal{M}(\mathbb{N} \times D)$ ; the data domain  $D$  is the set of tuples flowing in a particular channel. Notice that we need to have multisets because the same tuple under the same tag may appear in a channel more than one times. For example, the join processing node may produce many identical tokens. Consider now the two processing nodes, namely *join* and *dup-elim*. The join node takes as input two tag-extended relations; each such relation does not contain duplicates and is therefore an element<sup>2</sup> of  $\mathcal{P}(\mathbb{N} \times D)$  (which is a special case of  $\mathcal{M}(\mathbb{N} \times D)$ ). The output of the join is an element of  $\mathcal{M}(\mathbb{N} \times D)$ , since the joining of two relations may create duplicate tuples. Overall, the type of join is  $[\mathcal{P}(\mathbb{N} \times D)]^2 \rightarrow \mathcal{M}(\mathbb{N} \times D)$ . Consider now the *dup-elim* node, which eliminates duplicates from its input channel. Its type is  $\mathcal{M}(\mathbb{N} \times D) \rightarrow \mathcal{P}(\mathbb{N} \times D)$ . In fact, the dup-elim is nothing more than the function which given a multiset returns the corresponding underlying set.

### 4.3 Algorithms with Increased Asynchronicity

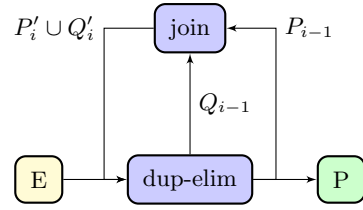
The transitive closure of an arbitrary relation can be calculated with the technique shown in the previous section, using a dataflow network that corresponds to an iterative Map-Reduce system. In this section, we calculate the transitive closure by *recursive doubling*, using a set of join and dup-elim nodes, as suggested in [2, 3]. In the form of pseudocode, the algorithm that we use (copied from [3]) is the following:

```

1:  $Q_0 := E$ 
2:  $P_0 := \{(x, x) \mid x \text{ is a graph node}\}$ 
3:  $i := 0$ 
4: repeat
5:    $i := i + 1$ 
6:    $P'_i(x, y) := \pi_{x,y}(Q_{i-1}(x, z) \bowtie P_{i-1}(z, y))$ 
7:    $Q'_i(x, y) := \pi_{x,y}(Q_{i-1}(x, z) \bowtie Q_{i-1}(z, y))$ 
8:    $P_i := P'_i \cup P_{i-1}$ 
9:    $Q_i := Q'_i - P_i$ 
10: until  $Q_i = \emptyset$ 

```

<sup>2</sup>By  $\mathcal{P}(A)$  we denote the power-set of a given set  $A$ .



**Figure 3:** The set of join and dup-elim nodes that computes the transitive closure by recursive doubling.

We deviate in the details of the implementation by introducing tags of the form  $(i, l)$  which are used to annotate every row  $(u, v)$ . A tag conveys two pieces of information:  $i$  is the number of iteration in which a row was produced, and  $l$  is the length of the path. Hashing can be used to determine which node will receive a given tagged row, in the same way as in [3]; we omit the details here.

Each *join node* has two inputs, left and right, and two distinct internal stores, again left and right. The join is performed as follows:

1. For every tuple  $(x, z)$  with tag  $(i_1, l_1)$  in its left input, the node searches its right store for tuples  $(z, y)$  with tag  $(i_2, l_2)$  such that  $l_2 \leq l_1$ . For every such tuple it emits a tuple  $(x, y)$  with tag  $(i, l)$  where  $i = \max\{i_1, i_2\} + 1$  and  $l = l_1 + l_2$ . The initial tuple  $(x, z)$  is stored in the left store together with its tag.
2. For every tuple  $(z, y)$  with tag  $(i_2, l_2)$  in its right input, the node searches its left store for tuples  $(x, z)$  with tag  $(i_1, l_1)$  such that  $l_2 \leq l_1$ . For every such tuple it emits a tuple  $(x, y)$  with tag  $(i, l)$  where  $i = \max\{i_1, i_2\} + 1$  and  $l = l_1 + l_2$ . The initial tuple  $(z, y)$  is stored in its right store together with its tag.

Each *dup-elim node* receive tuples  $(x, y)$  with tag  $(i, l)$  and performs the following steps:

1. If the tuple already exists with smaller tag (i.e., either smaller length or same length and smaller iteration number) then it is ignored. If the tag exists with bigger tag then the tag is updated to the smaller one and the tuple continues.
2. If  $l = 2^i$  then it is fed back both as left and as right input to the appropriate join nodes, otherwise it is fed only as right input.

Each tuple  $(x, y)$  in the relation  $E$  is sent to the corresponding dup-elim node with tag  $(0, 1)$ .

The left input of a join node corresponds to relation  $Q$  (the paths of size  $2^i$ ), whereas the right input corresponds to relation  $P$ . We choose to join each tuple of  $Q$  with every tuple of  $P$  of strictly smaller length. That join corresponds to line (6) of the recursive doubling algorithm. Moreover, we join each tuple of  $Q$  with every right tuple of equal length. That corresponds to line (7). Thus the output of the join is the union of both steps, but since we have tagged each tuple with the length of the path we can distinguish the two relations.

In fact, the dup-elim will pick only the tuples of size  $2^i$  and redirect them to the left input of the join. But since

$Q_i \subseteq P_{i+1}$  those tuples will also be redirected to the right input of the join. All the other tuples will be redirected only to the right input as part of  $P$ . Suppose now that a new tuple is generated with iteration number  $i$  and the same tuple already exists in dup-elim store with different tag. If the tag has a smaller iteration  $j$  then the tuple exists in  $P_j \subseteq P_i$ . In that case the new tuple is ignored. If that tuple is for the relation  $Q$ , omitting it corresponds to line (9) of the algorithm. On the other hand, if it was for the relation  $P$ , this corresponds to line (8). Since the iteration is not synchronous, it may occur that the new tuple has smaller tag (smaller length) than the one already stored in the dup-elim. In that case, the tuple is not ignored since it precedes in time and the node will emit the tuple as if it is encountered for the first time. It will also update the tag with the smaller one, to prevent for other duplicates to propagate.

Note that the relations  $Q$  and  $P$  up to iteration  $i$  are accumulated in the left and right stores of join nodes, respectively. Moreover, the relation  $P$  is also stored in the dup-elim nodes.

The dataflow is slightly different of that described in Subsection 4.2. In this case the tags are elements of  $\mathbb{N} \times \mathbb{N}$  and the channels are elements of  $\mathcal{M}((\mathbb{N} \times \mathbb{N}) \times D)$ . The type of join node is  $[\mathcal{P}((\mathbb{N} \times \mathbb{N}) \times D)]^2 \rightarrow \mathcal{M}((\mathbb{N} \times \mathbb{N}) \times D)$ . On the other hand, the type of the dup-elim node is  $\mathcal{M}((\mathbb{N} \times \mathbb{N}) \times D) \rightarrow [\mathcal{P}((\mathbb{N} \times \mathbb{N}) \times D)]^2$ . The dataflow graph is depicted in Figure 3. Note that the dup-elim node differs from the simple version since it has two output channels, one for each relation.

Consider the case where the structure of the graph changes, for example, new edges are added in a timely manner. A simple setup of the recursive doubling algorithm must restart the computation each time a change of the graph is detected. In the aforementioned tagged-dataflow graph a new edge will trigger the join of certain paths only, based on their tag, minimizing the redundant computations.

#### 4.4 Recursive Algorithms

To our knowledge, the existing extensions of Map-Reduce have mainly dealt with iterative algorithms. However, not all algorithms can be expressed elegantly in an iterative way; there exist problems whose solution is more naturally expressed in a recursive manner. In the rest of this subsection we consider one such problem and at the end of the subsection we discuss the possibility of extending the proposed technique to all recursively defined functions.

The algorithm we present below is a distributed sorting procedure that is based on the tagged-dataflow approach. However, we do not claim at present that this is actually an *efficient* way to perform sorting in a distributed environment nor that this example is fully-compatible with the nature of Map-Reduce. The algorithm described below is only intended to demonstrate that the tagging mechanism can also be used to implement *recursive algorithms* in a distributed manner. Intuitively, the tags can be used in order to keep track of the paths in the recursion tree.

Let us assume that we would like to sort a considerably large list of data, which is possibly distributed in a number of nodes over the network. One possibility is to use *merge sort*: we split the list into two parts, we sort each one of them separately and then merge the results. This is clearly a recursive procedure. The problem is therefore how we can

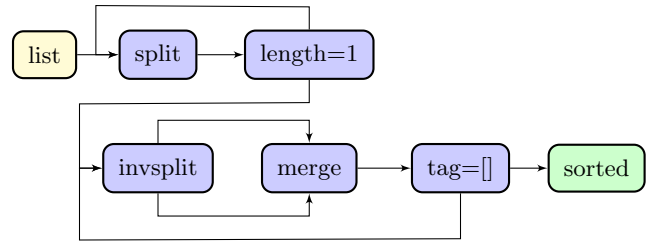


Figure 4: The mergesort depicted as dataflow.

implement the recursive *merge sort* in a distributed way, using tags to coordinate the whole process.

A *merge sort* function usually has the following form:

```
msort([]) = []
msort([x]) = [x]
msort(l) = merge(msort(l1), msort(l2))
where (l1, l2) = split(l)
```

The function `split` divides a list into two disjoint lists of half size and returns a tuple consisting of these two parts. The base cases of the recursion (`l == []` and `l == [x]`) need not be so fine-grained; for example, in a Map-Reduce context, when a list given to an intermediate step of merge sort is relatively small, we can sort it locally on a particular machine instead of continuing to divide it into smaller lists, which would circulate in the network and impose further communication costs. To simplify the presentation, we assume however that `msort` is defined as above and that the recursion reaches down to lists of trivial size.

The key idea behind the distributed implementation of merge sort is the following. We tag the all values in our list with two different tags. The first tag is a natural number (initially equal to 0) which will be used in the merge process of the algorithm. The second tag is a list of natural numbers (initially empty) which indicates the steps of the splitting procedure that have been applied on a particular element of the initial list. For example, if during the recursive execution of `msort` an element of the initial list was placed in the component `l1`, during the first invocation of `split`, and in the component `l2`, during the second invocation of `split`, then this element will be tagged by `[2, 1]`.

The distributed algorithm for merge sort is depicted in Figure 4. It contains two phases.

In the first phase, the algorithm starts by splitting the initial list into two sublists; this is achieved by the `split` node in Figure 4. The elements placed in the first component `l1` by `split` will have their list tag prefixed by 1, while those played in the second component `l2` will be prefixed by 2. This process of splitting and tagging is continued until all elements of the initial list have different tags; this is achieved by the `length=1` node, which checks whether the list of elements corresponding to a given tag contains a single element and, if not, returns all elements of the list to `split`. Every element whose tagging process has been completed, passes to the second phase of the dataflow network.

In the second phase of the algorithm, the `invsplit` node examines each element that arrives to it; if the head of the element's tag is 1, `invsplit` removes the head and sends the element to its left output; if the head is 2, it removes it and sends the element to the right output. The `merge` node sorts

all those elements that have the same list tag by changing the natural number tag so as to reflect the order of each element. E.g., if `merge` receives the element  $\langle\langle 0, [2, 1]\rangle, \text{George}\rangle$  and the element  $\langle\langle 0, [2, 1]\rangle, \text{Steve}\rangle$ , then, since `George` is alphabetically first compared to `Steve`, the `merge` node will output  $\langle\langle 0, [2, 1]\rangle, \text{George}\rangle$  and  $\langle\langle 1, [2, 1]\rangle, \text{Steve}\rangle$ . If in the next step it receives  $[\langle\langle 0, [1]\rangle, \text{Ann}\rangle, \langle\langle 1, [1]\rangle, \text{Suzan}\rangle]$  in the left input and  $[\langle\langle 0, [1]\rangle, \text{George}\rangle, \langle\langle 1, [1]\rangle, \text{Steve}\rangle]$  in the right input, then it will merge these two ordered sets and produce  $[\langle\langle 0, [1]\rangle, \text{Ann}\rangle, \langle\langle 1, [1]\rangle, \text{George}\rangle, \langle\langle 2, [1]\rangle, \text{Steve}\rangle, \langle\langle 3, [1]\rangle, \text{Suzan}\rangle]$ . The process will end when two ordered sets whose elements are all tagged with the empty list appear as inputs to `merge`; in this case, `merge` will do the final merging of these two sets, and the sorted file will come to the output.

The above procedure may seem ad hoc at first sight and one may assume that the distributed tag-based execution may not be applicable to all forms of recursion. However, this does not appear to be the case. A purely dataflow tag-based scheme for implementing first-order recursive functional programs was proposed in [26] and was theoretically justified in [22]. However, the scheme of [26] is appropriate for demand-driven execution while the main applications in the area of Map-Reduce appear to require a data-driven approach. The mergesort example presented above was obtained by adapting the technique of [26] to run in a data-driven way. Whether this can be done in general appears to be an interesting research problem.

## 5. FUTURE WORK

There exist several aspects of the connection between the Map-Reduce framework and tagged-dataflow that require further investigation. In the following, we outline certain such problems that we feel are particularly interesting and worthwhile for further study:

- Every functional dataflow network of the form presented in Section 3 can be shown to compute a function which is the least fixed point of a system of equations associated with the network. This result is an easy generalization of the so-called *Kahn principle* [18]. The least fixed point can be computed inductively, based on standard results of recursion theory (in particular, Kleene’s fixed point theorem). Therefore, there exists a way of computing the *meaning* of functional dataflow networks or, in other words, we have a formal theory for reasoning about functional dataflow programs. It would be interesting to examine whether the Kahn principle also holds for the more general (i.e., multiset-based) dataflow networks introduced in Subsection 4.1. If this is the case, then this opens up the possibility of performing formal proofs of correctness for various dataflow algorithms (such as the transitive closure algorithm of Section 4).
- At present, not many specialized programming languages have been proposed that can be used to program applications for the processing of massive data. We feel that this is probably one of the next steps in the evolution of the area. Such languages would definitely benefit by adopting the features and philosophy of the dataflow programming languages of the past.
- Based on the model of tagged-dataflow, one should be able to define a formal theory of fault tolerance for

dataflow networks. Since fault tolerance is a primary characteristic of (standard) Map-Reduce, it would be interesting to see how such a desired property can be ensured in the more general setting of the tagged-dataflow model.

We strongly believe that the further investigation of the interactions between dataflow and the novel approaches to distributed processing that have resulted from Map-Reduce, will prove very rewarding.

## 6. REFERENCES

- [1] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, editors, *Mathematical Foundations of Programming Semantics*, volume 442 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 1989.
- [2] F. N. Afrati, V. R. Borkar, M. J. Carey, N. Polyzotis, and J. D. Ullman. Cluster computing, recursion and datalog. In O. de Moor, G. Gottlob, T. Furche, and A. J. Sellers, editors, *Datalog*, volume 6702 of *Lecture Notes in Computer Science*, pages 120–144. Springer, 2010.
- [3] F. N. Afrati, V. R. Borkar, M. J. Carey, N. Polyzotis, and J. D. Ullman. Map-reduce extensions and recursive queries. In A. Ailamaki, S. Amer-Yahia, J. M. Patel, T. Risch, P. Senellart, and J. Stoyanovich, editors, *EDBT*, pages 1–8. ACM, 2011.
- [4] F. N. Afrati and J. D. Ullman. Transitive closure and recursive datalog implemented on clusters. In E. A. Rundensteiner, V. Markl, I. Manolescu, S. Amer-Yahia, F. Naumann, and I. Ari, editors, *EDBT*, pages 132–143. ACM, 2012.
- [5] Arvind and D. Culler. The tagged token dataflow architecture (preliminary version). Technical report, Laboratory for Computer Science, MIT, Cambridge, MA, 1983.
- [6] Arvind and R. S. Nikhil. Executing a program on the MIT tagged-token dataflow architecture. *IEEE Trans. Computers*, 39(3):300–318, 1990.
- [7] V. R. Borkar, M. J. Carey, R. Grover, N. Onose, and R. Vernica. Hyracks: A flexible and extensible foundation for data-intensive computing. In S. Abiteboul, K. Böhm, C. Koch, and K.-L. Tan, editors, *ICDE*, pages 1151–1162. IEEE Computer Society, 2011.
- [8] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. HaLoop: Efficient iterative data processing on large clusters. *PVLDB*, 3(1):285–296, 2010.
- [9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *NSDI*, pages 313–328. USENIX Association, 2010.
- [10] A. L. Davis. The architecture and system method of DDM1: A recursively structured data driven machine. In E. J. McCluskey, J. F. Wakerly, E. D. Crockett, T. E. Bredt, D. J. Lu, W. M. van Cleemput, S. S. Owicki, R. C. Ogus, R. Apte, M. D. Beaurdy, and J. Losq, editors, *ISCA*, pages 210–215. ACM, 1978.
- [11] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

- [12] J. B. Dennis and D. Misunas. A preliminary architecture for a basic data flow processor. In W. K. King and O. N. Garcia, editors, *ISCA*, pages 126–132. ACM, 1974.
- [13] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica. Shark: fast data analysis using coarse-grained distributed memory. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *SIGMOD Conference*, pages 689–692. ACM, 2012.
- [14] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *PVLDB*, 5(11):1268–1279, 2012.
- [15] J. R. Gurd, C. C. Kirkham, and I. Watson. The Manchester prototype dataflow computer. *Commun. ACM*, 28(1):34–52, 1985.
- [16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, pages 59–72, 2007.
- [17] R. Jagannathan, C. Dodd, and I. Agi. GLU: A high-level system for granular data-parallel programming. *Concurrency - Practice and Experience*, 9(1):63–83, 1997.
- [18] G. Kahn. The semantics of simple language for parallel programming. In *IFIP Congress*, pages 471–475, 1974.
- [19] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SPAA*, page 48, 2009.
- [20] F. McSherry, D. G. Murray, R. Isaacs, and M. Isard. Differential dataflow. In *CIDR*. [www.cidrdb.org](http://www.cidrdb.org), 2013.
- [21] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas. MRShare: Sharing across multiple queries in mapreduce. *PVLDB*, 3(1):494–505, 2010.
- [22] P. Rondogiannis and W. W. Wadge. First-order functional languages and intensional logic. *J. Funct. Program.*, 7(1):73–101, 1997.
- [23] R. D. Tennent. *Principles of programming languages*. Prentice Hall International Series in Computer Science. Prentice Hall, 1981.
- [24] W. W. Wadge and E. A. Ashcroft. *LUCID, the Dataflow Programming Language*. Academic Press Professional, Inc., San Diego, CA, USA, 1985.
- [25] I. Watson and J. R. Gurd. A prototype data flow computer with token labelling. In *Proceedings of the National Computer Conference*, pages 623–628, 1979.
- [26] A. A. Yaghi. *The Intensional Implementation Technique for Functional Languages*. PhD thesis, Department of Computer Science, University of Warwick, Coventry, UK, 1984.
- [27] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: fault-tolerant streaming computation at scale. In M. Kaminsky and M. Dahlin, editors, *SOSP*, pages 423–438. ACM, 2013.



# Processing Regular Path Queries on Giraph

Maurizio Nolé  
DIMIE - Università della Basilicata  
Via dell'Ateneo Lucano 10  
Potenza, Italy  
mnole@gmail.com

Carlo Sartiani  
DIMIE - Università della Basilicata  
Via dell'Ateneo Lucano 10  
Potenza, Italy  
sartiani@gmail.com

## ABSTRACT

In the last few years social networks have reached an ubiquitous diffusion. Facebook, LinkedIn, and Twitter now have billions of users, that daily interact together and establish new connections. Users and interactions among them can be naturally represented as *data graphs*, whose vertices denote users and whose edges are labelled with information about the different interactions.

In this paper we sketch a novel approach for processing regular path queries on very large graphs. Our approach exploits Brzozowski's derivation of regular expressions to allow for a vertex-centric, message-passing-based evaluation of path queries on top of Apache Giraph.

## Categories and Subject Descriptors

H.2.4 [Database Management]: Systems

## General Terms

Algorithms, Performance

## Keywords

Graph Query Processing, Distributed Computing

## 1. INTRODUCTION

In the last few years social networks have reached an ubiquitous diffusion. Facebook, LinkedIn, and Twitter now have billions of users, that daily interact together and establish new connections. Users and their interactions can be naturally represented as *data graphs*, whose vertices denote users and whose edges are labelled with information about the different interactions. The problem of managing, querying, and mining graph databases, hence, is becoming more and more important; similar problems emerge in many different application fields where data have a graph structure, e.g., traffic analysis, crime detection, the Semantic Web.

Data graphs have attracted a significant research interest since the mid 90's. In particular, several query languages

based on regular expressions [8, 9, 6] have been proposed and a few data graph query processors have been designed [5, 10].

Today data graphs can be very large and easily exceed 100 millions vertices. To manage this kind of graphs, Google designed a novel class of graph processing systems, based on the Bulk Synchronous Parallel Model by Valiant [11], where graphs are automatically partitioned across the nodes of a computing cluster, and algorithms are expressed through *vertex-centric* functions, i.e., functions that are executed by each vertex in the graph. Systems in this class (e.g., Google Pregel [7] and Apache Giraph [1]) exhibit very good scalability properties for many graph algorithms, but have not been designed for querying graphs.

*Our Contribution.* In this paper we sketch a novel approach for evaluating path queries on very large graphs. Our approach exploits Brzozowski's derivation of regular expressions [4] to allow for a vertex-centric, message-passing-based evaluation of path queries on top of Giraph (see Section 3.1). In particular, when each vertex receives a query  $q$ , it derives  $q$  according to the symbols labelling outgoing edges and propagates the derivative of  $q$  to its neighbours. To avoid network flooding, only outgoing edges labelled with symbols in the *first set* of  $q$  are considered.<sup>1</sup>

## 2. DATA MODEL AND QUERY LANGUAGE

### 2.1 Data Model

Following [6], we model a *data graph* as an edge-labelled graph, as shown below.

**Definition 2.1 (Data Graph)** *Given a finite alphabet  $\Sigma$  and a (possibly) infinite value domain  $\mathcal{D}$ , a data graph  $G$  over  $\Sigma$  and  $\mathcal{D}$  is a triple  $G = (V, E, \rho)$ , where:*

- $V$  is a finite set of vertices;
- $E \subseteq V \times \Sigma \times V$  is a set of labelled, directed edges  $(v_i, a, v_j)$ ;
- $\rho : V \rightarrow \mathcal{D}$  is a mapping from vertices to values.

Given a vertex  $v$ , we will indicate with  $in(v)$  and  $out(v)$  the set of incoming and outgoing edges, respectively. More formally:

<sup>1</sup>The first set of a regular expression  $r$  is the set of symbols that appear in the first position of words matching  $r$ .



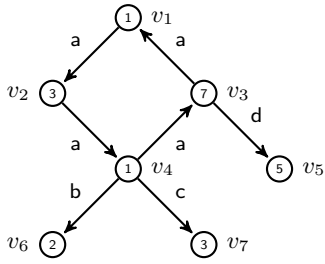


Figure 1: A graph.

- $in(v) = \{(v', a, v) \in E \mid v' \in V \wedge a \in \Sigma\}$ ;
- $out(v) = \{(v, a, v') \in E \mid v' \in V \wedge a \in \Sigma\}$ .

We will also indicate with  $id(v)$  the unique identifier of  $v$ . We assume that sequences of outgoing (incoming) edges of a vertex are unordered, as it is often the case in graph databases.

## 2.2 Query Language

Many query languages for graph data have been presented in the past [8, 9, 6]. We focus our attention here on GXPath, a language recently proposed by Libkin et al. in [6]. GXPath is based on the idea of using regular expressions to specify patterns that must be matched by paths in the input graph. Given a query  $q$ , the result of its evaluation over a graph  $G$  is always a set of vertex pairs  $(v, v')$  such that  $v$  and  $v'$  are connected by a path  $p$  in  $G$  matching the query  $q$ . GXPath extends other path languages like RPQs or NREs with the introduction of the *complement* operator, data tests on the values stored into vertices, as well as counters, which generalize the Kleene star, and it can be considered as an adaptation of XPath [2] to data graphs.

Among the various fragments of GXPath, we focus here on a navigational, path-positive fragment with counting, but without complement, intersection, and nested conditions, as described by the following grammar.

$$\alpha ::= \epsilon \mid \_ \mid a \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha^{m,n}$$

Given a graph  $G = (V, E, \rho)$ , the semantics of our fragment of GXPath can be defined as follows.

$$\begin{aligned} \llbracket \epsilon \rrbracket_G &= \{(u, u) \mid u \in V\} \\ \llbracket \_ \rrbracket_G &= \{(u, v) \mid \exists a \in \Sigma. (u, a, v) \in E\} \\ \llbracket a \rrbracket_G &= \{(u, v) \mid (u, a, v) \in E\} \\ \llbracket \alpha_1 + \alpha_2 \rrbracket_G &= \llbracket \alpha_1 \rrbracket_G \cup \llbracket \alpha_2 \rrbracket_G \\ \llbracket \alpha_1 \cdot \alpha_2 \rrbracket_G &= \llbracket \alpha_1 \rrbracket_G \circ \llbracket \alpha_2 \rrbracket_G \\ \llbracket \alpha^{m,n} \rrbracket_G &= \cup_{i=m}^n \llbracket \alpha \rrbracket_G^i \end{aligned}$$

where  $\circ$  is the symbol for the concatenation of binary relations and  $R^i$  denotes the concatenation of  $R$  with itself  $i$  times. Here,  $\epsilon$  denotes the empty word,  $\_$  matches any symbol,  $\alpha_1 \cdot \alpha_2$  and  $\alpha_1 + \alpha_2$  are the standard concatenation and union operators, and  $\alpha^{m,n}$  denotes the repetition of  $\alpha$  from  $m$  to  $n$  times ( $m \in \mathbb{N}$ ,  $n \in \mathbb{N} \cup \{*\}$ ,  $m \leq n$ ).

**Example 2.2** Consider the graph depicted in Figure 1.

Consider now the following query:  $a^{2,3} \cdot (b+d)$ . This query returns all vertex pairs  $(u, v)$  connected by the following paths:  $aab$ ,  $aaab$ ,  $aad$ ,  $aaad$ . The result of this query is  $\{(v_1, v_6), (v_1, v_5), (v_2, v_5), (v_5, v_6)\}$ .

## 3. PROCESSING PATH QUERIES

### 3.1 Brzowski's derivatives

Brzowski's derivatives [3] represent an alternative way to check if a word  $w$  belongs to the language generated by a given regular expression  $r$ . The idea is to iterate over  $w$  and to rewrite  $r$  according to the last read symbol, hence computing a *derivative*; if the derivative generated after the last symbol of  $w$  has been read is  $\epsilon$ , then the check is successful.

Brzowski's derivatives can be extended to regular path expressions on data graphs in the following way.

**Definition 3.1 (Derivative)**  $\alpha'$  is a derivative of  $\alpha$  in a graph  $G = (V, E)$  according to  $a \in \sigma$  iff  $\bigcup_{(u,a,v) \in E} \{(u, v)\} \circ \llbracket \alpha' \rrbracket_G = \llbracket \alpha \rrbracket_G$ .

**Definition 3.2 (Empty expression)**  $\emptyset$  denotes the empty regular expression, that is,  $\llbracket \emptyset \rrbracket_G =_{def} \emptyset$

**Proposition 3.3 (Empty expression properties)**  $\emptyset$  satisfies the following properties:

$$\begin{aligned} \alpha + \emptyset &= \emptyset + \alpha = \alpha \\ \alpha \cdot \emptyset &= \emptyset \cdot \alpha = \emptyset \end{aligned}$$

**Notation 3.4** ( $m^-$ ,  $*-1$ ) In the following definitions, we use  $m^-$  to denote  $\max(m-1, 0)$ , and assume that  $*-1 = *$ .

**Definition 3.5**  $N(\alpha)$  is a predicate on regular expressions, defined as follows:

$$\begin{aligned} N(\emptyset) &= \text{false} & N(\epsilon) &= \text{true} \\ N(a) &= \text{false} & N(\_) &= \text{false} \\ N(\alpha_1 + \alpha_2) &= N(\alpha_1) \vee N(\alpha_2) \\ N(\alpha_1 \cdot \alpha_2) &= N(\alpha_1) \wedge N(\alpha_2) \\ N(\alpha^{m,n}) &= N(\alpha) \end{aligned}$$

**Definition 3.6**  $first(\alpha)$  is a function on regular expressions, defined as follows:

$$\begin{aligned} first(\emptyset) &= \emptyset \\ first(\epsilon) &= \emptyset \\ first(a) &= \{a\} \\ first(\_) &= \Sigma \\ first(\alpha_1 + \alpha_2) &= first(\alpha_1) \cup first(\alpha_2) \\ first(\alpha_1 \cdot \alpha_2) &= \begin{cases} first(\alpha_1) \cup first(\alpha_2) & \text{if } N(\alpha_1) \\ first(\alpha_1) & \text{otherwise} \end{cases} \\ first(\alpha^{m,n}) &= first(\alpha) \end{aligned}$$

**Definition 3.7 (Derivation)**  $d_a(\alpha)$ , where  $\alpha$  is a regular path query and  $a \in \Sigma$ , is defined in Figure 2.

It is easy to see that  $d_a(\alpha)$  is a derivative of  $\alpha$  according to  $a$  in  $G$ .

### 3.2 Evaluation Algorithm

Brzowski's derivatives can be used to implement path query processing on top of Giraph (or similar systems). In Giraph a computation consists of several *supersteps*, representing global synchronization points. Each superstep comprises a *master* computation, performed by a special node ("the master") at the beginning of the superstep, and by a

$$\begin{aligned}
d_a(\epsilon) &=_{def} \emptyset \\
d_a(\emptyset) &=_{def} \emptyset \\
d_a(b) &=_{def} \begin{cases} \epsilon & \text{if } a = b \text{ or } b = \_ \\ \emptyset & \text{otherwise} \end{cases} \\
d_a(\alpha_1 + \alpha_2) &=_{def} d_a(\alpha_1) + d_a(\alpha_2) \\
d_a(\alpha_1 \cdot \alpha_2) &=_{def} \begin{cases} d_a(\alpha_1) \cdot \alpha_2 + d_a(\alpha_2) & \text{if } N(\alpha_1) \\ d_a(\alpha_1) \cdot \alpha_2 & \text{otherwise} \end{cases} \\
d_a(\alpha^{m,n}) &=_{def} \begin{cases} \emptyset & \text{if } n = 0 \\ d_a(\alpha) \cdot \alpha^{m^-,n-1} & \text{if (not } N(\alpha) \text{ and } n > 0) \text{ or } (m = 0 \text{ and } n = *) \\ d_a(\alpha) \cdot \alpha^{m^-,n-1} + d_a(\alpha^{m^-,n-1}) & \text{otherwise} \end{cases}
\end{aligned}$$

Figure 2: Derivation function.

*vertex-centric* computation, where each graph vertex processes incoming messages from other vertices, sends messages to other vertices, and executes a given algorithm. Vertices communicate together through messages, that are delivered at the beginning of the next superstep; the communication between the master and vertices is performed through special data structures called *aggregators*, and is bidirectional. The computation halts when each vertex decides to halt and no more message is flowing in the network.

In our system each input query  $q$  is sent to each vertex by the master at the beginning of the first superstep; the master also sends the command **derive**. If  $q$  contains a subexpression of the form  $\alpha^{m,*}$ , the master transforms it into  $\alpha^{m,|V|}$

At superstep 0, each vertex  $v$  checks for a message by the master; if the message is a pair  $(q, \mathbf{derive})$ , then the vertex is instructed to start the derivation process. If  $q = \epsilon$  or  $N(q)$ , then  $v$  sends the pair  $(id(v), id(v))$  to the master through the aggregator **result**; the master will push the content of **result** on persistent store at the beginning of the next superstep. If  $q \neq \emptyset$  and  $q \neq \epsilon$ ,  $v$  starts the derivation process by looking at the symbols labelling outgoing edges. The derivation of  $q$  according to a symbol  $a$  is performed only if  $a \in first(q)$ , i.e.,  $a$  may appear in the first position of a word generated by  $q$ . If  $d_a(q) \neq \emptyset$ , then  $v$  sends to the target vertex a message  $(id(v), d_a(q))$ .

At each superstep  $s > 0$ , the master checks if the new results have been aggregated in **result** by graph vertices in the previous superstep; in that case, the master moves the result pairs to the persistent store and cleans the aggregator.

At each superstep  $s > 0$ , each vertex  $v$  checks if there are incoming messages from other vertices. If  $v$  receives a message  $m = (id(v_0), q')$ , then it starts analyzing  $q'$  to understand if it must be derived; in particular, if  $q' = \epsilon$  or  $N(q')$ ,  $v$  adds the pair  $(id(v_0), id(v))$  to **result** and, if  $q' = \epsilon$ , it starts processing the next message. If  $q' \neq \epsilon$ ,  $v$  starts deriving  $q'$  according to the symbols labelling outgoing edges, provided that they are in  $first(q')$ . If the derivative is equal to  $\emptyset$ , no message is sent; otherwise, given an edge  $(v, a, v')$ ,  $v$  sends to  $v'$  the message  $(id(v_0), d_a(q'))$ .

The pseudocode of the algorithms for master and vertex computation is shown in Figures 3 and 4.

### 3.3 Implementation Issues

We implemented the algorithms described in the previous section in a very preliminary research prototype called

```

MASTERCOMPUTE
/ - - Input: a query q
/ - - Input: aggregators result and command
1 if (superstep == 0)
2   if (q == C[αm,*]) q = C[αm,|V|]
3   command.aggregate((q, derive))
4 else List resultList = result.getAggrValue()
5   if (resultList ≠ {})
6     add resultList to persistent storage
7     result.clean()

```

Figure 3: Master computation.

Vertigo. While implementing Vertigo, we faced several challenges. First of all, Brzozowski’s derivation does not behave well on non-deterministic regular expressions, as it may generate exponentially larger derivatives;<sup>2</sup> this is even more evident when the regular expression contains a counting operator. To speed up the derivation process, our derivation algorithm works modulo associativity and commutativity of union. In detail, we memoize the derivation process through a small LRU cache on each Giraph worker, and systematically simplify derivatives through the following rules:  $\alpha + \alpha \rightarrow \alpha$ ,  $\alpha + \emptyset \rightarrow \alpha$ ,  $\alpha \cdot \epsilon \rightarrow \alpha$ ,  $\alpha \cdot \emptyset \rightarrow \emptyset$ .

Second, when working on very large graphs and queries with low selectivity, query results can be quite large. Hence, their transmission to the master through aggregators can be quite expensive. To decrease this overhead, result transmission is performed at the end of each superstep by each worker through a post-superstep computation. Workers perform a preliminary duplicate elimination, while the master just appends result pairs on a file on HDFS; final duplicate elimination is performed when computation halts.

Finally, the most efficient way to evaluate queries of the form  $\alpha^{m,*}$  is to evaluate  $\alpha$  and, then, to compute the reflexive and transitive closure of  $\llbracket \alpha \rrbracket_G$ . This can be performed in a BSP fashion, but it would be too expensive as it requests to stop the current graph traversal. Therefore, we prefer to drop this technique in favour of a more naive one, where \*

<sup>2</sup>Intuitively, a regular expression  $r$  is non-deterministic (or 1-ambiguous) if there exists at least a word  $w$  that can match  $r$  in multiple ways.

is replaced by the number of vertices in the input graph.

## 4. CONCLUSIONS AND FUTURE WORK

In this paper we sketched a novel algorithm for evaluating regular path queries on data graphs. This algorithm exploits Brzozowski’s derivation and can be used in Giraph and any other similar system. We developed a very preliminary prototype implementation of our algorithm; in early tests on a single commodity machine this prototype easily processed queries on 200-million-edge graphs. We are currently testing our implementation on Pivotal’s AWB cluster.

In a very near future we plan to extend our algorithm to support a larger fragment of GXPath comprising backward navigation, branching, and intersection.

## 5. ACKNOWLEDGMENTS

(Portions of) the research in this paper use results obtained from the Greenplum Analytics Workbench, made available by Greenplum, a division of GoPivotal Corporation.

## 6. REFERENCES

- [1] Apache giraph, 2013. <http://http://giraph.apache.org>.
- [2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0 (Second Edition). Technical report, World Wide Web Consortium, 2010. W3C Recommendation.
- [3] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Inf. Comput.*, 142(2):182–206, 1998.
- [4] J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.
- [5] A. Koschmieder and U. Leser. Regular path queries on large graphs. In A. Ailamaki and S. Bowers, editors, *SSDBM*, volume 7338 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2012.
- [6] L. Libkin, W. Martens, and D. Vrgoc. Querying graph databases with XPath. In W.-C. Tan, G. Guerrini, B. Catania, and A. Gounaris, editors, *ICDT*, pages 129–140. ACM, 2013.
- [7] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In A. K. Elmagarmid and D. Agrawal, editors, *SIGMOD Conference*, pages 135–146. ACM, 2010.
- [8] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM J. Comput.*, 24(6):1235–1258, 1995.
- [9] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.
- [10] M. Sarwat, S. Elnikety, Y. He, and G. Kliot. Horton: Online query execution engine for large distributed graphs. In A. Kementsietsidis and M. A. V. Salles, editors, *ICDE*, pages 1289–1292. IEEE Computer Society, 2012.
- [11] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

```

VERTEXCOMPUTE
  / - - Input: aggregators result and command
  / - - Input: current vertex v
1  if (superstep == 0)
2      MasterMessage m = command.getAggrValue()
3      Query q = m.query
4      if (q ==  $\emptyset$ ) HALT()
5      elseif (q ==  $\epsilon$ )
6          result.aggregate((id(v), id(v)))
7          HALT()
8      elseif (N(q)) result.aggregate((id(v), id(v)))
9      for each (v, a, u)  $\in E$  : (a  $\in$  first(q))
10         Query q' = da(q)
11         if (q'  $\neq \emptyset$ )
12             if (q' == q'1 + q'2)
13                 if (q'1  $\neq \emptyset$ ) SEND(u, (id(v), q'1))
14                 if (q'2  $\neq \emptyset$ ) SEND(u, (id(v), q'2))
15             else SEND(u, (id(v), q'))
16             HALT()
17 elseif (superstep > 0)
18     for each message m = (id(v0), q)
19         if (q ==  $\epsilon$ )
20             result.aggregate((id(v0), id(v)))
21             skip to next message
22         elseif (N(q)) result.aggregate((id(v0), id(v)))
23         for each (v, a, u)  $\in E$  : (a  $\in$  first(q))
24             Query q' = da(q)
25             if (q'  $\neq \emptyset$ )
26                 if (q' == q'1 + q'2)
27                     if (q'1  $\neq \emptyset$ )
28                         SEND(u, (id(v0), q'1))
29                     if (q'2  $\neq \emptyset$ )
30                         SEND(u, (id(v0), q'2))
31                 else SEND(u, (id(v0), q'))
32                 HALT()

```

Figure 4: Vertex computation.

# Graph-Parallel Entity Resolution using LSH & IMM

Pankaj Malhotra  
TCS Innovation Labs, Delhi  
Tata Consultancy Services  
Ltd., Sector 63  
Noida, Uttar Pradesh, India  
malhotra.pankaj@tcs.com

Puneet Agarwal  
TCS Innovation Labs, Delhi  
Tata Consultancy Services  
Ltd., Sector 63  
Noida, Uttar Pradesh, India  
puneet.a@tcs.com

Gautam Shroff  
TCS Innovation Labs, Delhi  
Tata Consultancy Services  
Ltd., Sector 63  
Noida, Uttar Pradesh, India  
gautam.shroff@tcs.com

## ABSTRACT

In this paper we describe graph-based parallel algorithms for entity resolution that improve over the map-reduce approach. We compare two approaches to parallelize a Locality Sensitive Hashing (LSH) accelerated, Iterative Match-Merge (IMM) entity resolution technique: BCP, where records hashed together are compared at a single node/reducer, vs an alternative mechanism (RCP) where comparison load is better distributed across processors especially in the presence of severely skewed bucket sizes. We analyze the BCP and RCP approaches analytically as well as empirically using a large synthetically generated dataset. We generalize the lessons learned from our experience and submit that the RCP approach is also applicable in many similar applications that rely on LSH or related grouping strategies to minimize pair-wise comparisons.

## 1. MOTIVATION AND INTRODUCTION

The map-reduce (MR) parallel programming paradigm [9] and its implementations such as Hadoop [24] have become popular platforms for expressing and exploiting parallelism due to the ease with which parallelism can be abstracted to a higher-level. However, it has become increasingly apparent that there are classes of algorithms for which MR may not be well suited, such as those involving iterative or recursive computation. Graph-based parallelism via the Pregel programming paradigm [19] and its various implementations such as Giraph [2], Graphlab [18], or GPS [22] is an alternative approach that has been shown to perform better in such scenarios, for example in sparse-matrix multiplications, page-rank calculation, or shortest-paths in graphs etc.

In this paper we focus on the entity resolution (ER) problem and submit that graph-based parallelism is better suited for it. We consider the iterative match-merge (IMM) approach to ER [4], accelerated by locality-sensitive hashing (LSH) [1] to avoid unnecessary comparisons. In this context we found that the strategy that is natural if using MR, i.e., where records hashed to the same bucket are compared at a single reducer / node, need not be the most efficient approach. Instead, the graph-parallel model offers an alternative mechanism that is better at distributing the computational load across processors.

We introduce the IMM+LSH approach for ER later in this section. Next, in Section 2, we discuss how this entity-resolution algorithm should be parallelized, via MR as well as its natural translation to the graph-parallel model, which we call bucket-centric parallelization (BCP). We then describe our alternative technique for record-centric parallelization (RCP). In Section 3, we analyze the BCP and RCP approaches analytically as well as empirically using a large synthetically generated dataset.

We believe the lessons learned from this exercise of parallelizing ER are more general. The RCP approach appears better suited at dealing with the skewed work-loads that naturally arise when items need to be compared efficiently using probabilistic hashing or executing similarity joins. Further, when records containing large attributes (such as documents or images) need to be matched, albeit even exactly as in standard multi-way joins, our experience here leads us to suggest mechanisms to avoid unnecessary communication of large attributes that do not figure in the final matching result. We present these learnings in Section 5, after describing related work in Section 4.

### 1.1 Entity Resolution via LSH and IMM

We assume that information about real-world entities is available from disparate data sources in the form of records, with each record (such as a passport, or driving license) belonging to a unique real-world entity. Two records are said to *match* if a suitable match function returns ‘true’ indicating that the records belong to the same entity. Match functions can be implemented in different ways, such as rules, or even binary classifiers derived via machine learning. Under certain conditions [4] matching records may be *merged* to produce a new record that derives its attributes and values (e.g. via a union) from the matching input records.

Given a collection of records where each record belongs to a unique entity, **ER** seeks to determine disjoint subsets of records where the records in a subset match under some match function and form an entity by merging these records. For example, different records belonging to the same person, such as voter card, passport, and driving-licence should get merged to form one entity.

The R-Swoosh IMM algorithm as described in [4], performs ER as follows: Initialise the set  $I$  to contain all records and the set  $I'$  to be empty. R-Swoosh iterates over the records in set  $I$ , removing a record  $r$  from  $I$  and comparing it to records in  $I'$ . As soon as a matching record  $r'$  is found in  $I'$ , it is removed from  $I'$ , and a new record obtained by merging  $r$  and  $r'$  is added to  $I$ . On the other hand, if no matching record is found in  $I'$ , the record  $r$  is added to  $I'$ . The procedure continues until the set  $I$  is empty and  $I'$  contains a set of merged records representing the resolved entities.

The time complexity of IMM is quadratic in the number of records to be resolved, so it makes sense to attempt to pre-group records so that records in different groups are highly unlikely to

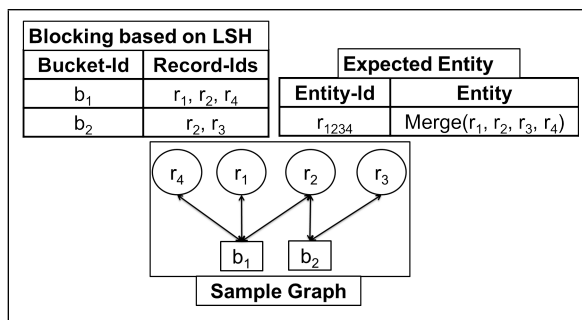


Figure 1: Sample Data and Corresponding Graph

match with each other. One way to achieve such grouping is via LSH [1]. LSH hashes each record to *one or more buckets* so that records that do not share a bucket are highly unlikely to match [20]. Therefore, instead of performing IMM on the entire set of records, only records belonging to the same bucket are considered for IMM.

## 2. PARALLEL ENTITY RESOLUTION

### 2.1 Bucket-centric Approach

#### 2.1.1 BCP using Map-Reduce

We use standard LSH via minhashing [5] to create buckets of potentially similar records. A Match function implementation may not use all the attributes in records to compare them. In other words, not all attributes in records may be relevant for the purpose of comparison of records. We use only the words occurring in the values of the relevant attributes for the purpose of hashing also. Each such relevant word occurring in any of the records is mapped to a unique integer. For each record, we consider the set of integers  $S$  corresponding to the set of relevant words it contains. The minhash for the set  $S$  is calculated as the  $\min((sx_i + z) \bmod Q)$ ,  $\forall x_i \in S$  (where  $s$  and  $z$  are random integers, and  $Q$  is a large prime). Different combinations of  $s$  and  $z$  determine different hash functions. For each record, we compute  $a \times b$  hash values using  $a \times b$  minhash functions. Out of these hash-values of a record,  $a$  are concatenated to get a *bucket-id*, we therefore get a total of  $b$  such bucket-ids for each record. Finally, a bucket consists of the record-ids of all the records that get hashed to the corresponding bucket-id.

A natural way to execute the above procedure in parallel using MR is to generate  $b$  key-value pairs [*bucket-id*, *record*] for every record in the map phase [8]. Records mapped to the same bucket-id are sent to a common reducer where IMM is run for each bucket. The result of IMM in each bucket is a set of ‘partially’ resolved entities since the possibility remains for records belonging to a single entity to get mapped to more than one bucket.

Consider the example shown in Figure 1, we have a collection  $\mathcal{R}$  of four records:  $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$  such that all the records belong to the same entity  $r_{1234}$ . The match function applied to any pair of records in  $\mathcal{R}$  gives `true`. Assuming  $a = 1$  and  $b = 3$ , each of the 4 records is hashed to 3 buckets using LSH. Of the buckets generated by LSH on the records in  $\mathcal{R}$ , only two buckets  $b_1$  and  $b_2$  end up with more than one record being hashed to them. The *singleton buckets*, i.e., the buckets having only one record hashed to them are not shown in the Figure 1. Therefore, bucket to record mapping becomes  $\{b_1, \{r_1, r_2, r_4\}\}, \{b_2, \{r_2, r_3\}\}$ . As a result of IMM on  $b_1$ , we get a partial-entity  $e_{b_1}$  consisting of  $\{r_1, r_2, r_4\}$ . Similarly, IMM on  $b_2$  gives another partial-entity  $e_{b_2}$  consisting of

$\{r_2, r_3\}$ . The ‘partial-entities’  $e_{b_1}$  and  $e_{b_2}$  belong to the same entity since they contain an overlapping record  $r_2$ . We consolidate the *partial-entities* emerging from all the buckets by computing connected components in a graph of records (similar to [23]), where an edge exists between two records if they belong to the same partial-entity, as shall be explained later. If a pair of partial-entities  $e_a$  and  $e_b$  happen to share at least one of the original records  $r_i$ , they end up being in the same connected component and all the records in them get merged.

The above approach using MR has two potential problems: First, a large number of buckets are singletons, so many reduce keys get only one record as a value. Such records do not need to be compared with any other record, so sending them to the reducers is unnecessary and causes significant communication overhead especially when records are large in size. Secondly, we need to find connected components after the first MR phase, which is itself an iterative procedure and is likely to perform better using the Pregel model.

#### 2.1.2 BCP using Pregel

Consider using MR to generate LSH buckets and discard singletons, and a graph-parallel approach using the Pregel paradigm thereafter. A high-level block diagram of how this works is shown in Figure 2. We perform LSH using MR as earlier except that instead of passing the records themselves we ensure that mappers only send record-ids, thus significantly reducing communication costs. In the reduce phase, instead of running IMM we merely generate the adjacency-list of a graph with two types of vertices, a *record-vertex* for each record in the collection and a *bucket-vertex* for each bucket obtained through LSH for collection of records, along with edges between them as follows: A *record-vertex* has outgoing edges to all the bucket-vertices corresponding to the buckets it gets hashed to. A *bucket-vertex* has outgoing edges to all the record-vertices which are hashed to it.

Note that since reducers know the size of every bucket they are responsible for, singletons are easily removed at this stage itself, i.e., *no* vertices are created for singleton buckets. As a result, only buckets containing record-ids that need to be compared are passed on to subsequent stages, eliminating the need to ship record contents for records in singleton buckets.

Note also that edges in the resulting graph are bi-directional, i.e., if there is an edge from  $v$  to  $v'$  then there is also an edge between  $v'$  and  $v$ . The adjacency-list files created by MR and also files mapping record-ids to records are inputs to a graph-parallel Pregel-based platform (such as Apache Giraph [2]), so that record-vertices are initialised with both record-ids as well as the full content of each record. Thereafter graph-parallel computations in the Pregel-model proceed via a number of supersteps as follows:

**SS1:** Each bucket-vertex is to perform IMM on all records that are hashed to it. Initially, only the IDs of the records hashed to a bucket are available at bucket-vertex via its outgoing edge-list. So in this superstep each record-vertex sends its value (which includes the record’s content) to the bucket-vertices in its outgoing edge-list (which is the list of all the non-singleton buckets the record is hashed to). For example, for the graph in Figure 1, record-vertex  $r_2$  sends its value to  $b_1$  and  $b_2$ .

**SS2:** Bucket-vertices receive the contents of all the records hashed to them, after which records at each bucket-vertex are compared using IMM. The result is a set of merged records or *partial-entities* at each bucket-vertex.

A bucket-vertex randomly selects one of the records in each partial-entity (merged record) as a *central record* for that partial-entity. Next the bucket-vertex sends a *graph-mutation* request so

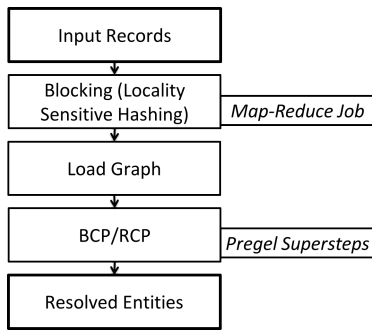


Figure 2: Overall flow across Map-reduce and Pregel

as to create a bi-directional edge between vertices corresponding to the central record and each of the remaining records of the partial-entity. As a result, all the record-vertices involved in a partial-entity get *connected* to each other through the vertex corresponding to the central record.

For the example in Figure 1, the records  $r_1$ ,  $r_2$ , and  $r_4$  are merged to give a partial-entity  $r_{124}$  at bucket-vertex  $b_1$ . Assuming  $r_1$  to be the central record, graph-mutation requests are sent by  $b_1$  to create bi-directional edges between  $r_1$  and  $r_2$ , and  $r_1$  and  $r_4$ . Similarly, at bucket-vertex  $b_2$ , we get a partial-entity  $r_{23}$  where a graph-mutation request is sent to create edges between  $r_2$  and  $r_3$ .

**SS3cc:** As a result of the previous superstep we know that there is a path connecting records belonging to a single entity even though they may have been part of different partial-entities resolved at different buckets. So we now find the connected components in the part of the graph consisting only of the record-vertices and the edges between them, ignoring the bucket-vertices and their edges. Note that it may take more than one superstep to find connected-components in this graph. Finding connected-components in a graph using the Pregel model is straightforward, so we omit its details for brevity.

Finally every record-vertex gets a connected-component id which corresponds to the entity it is resolved to. For the example in Figure 1, we will get one connected-component containing all the 4 records  $r_1$ ,  $r_2$ ,  $r_3$ , and  $r_4$ . It remains a simple matter to exchange component information between record-vertices to complete the resolution process: Record-vertices mutate the graph to create entity-vertices corresponding to their component-id along with edges to these; the platform ensures that only one instance of each distinct entity-vertex is created, with the result that all records for that entity are now connected to it. Records send themselves to their entity-vertex where final merging takes place to produce a resolved entity.

While the above approach using both MR and Pregel avoids shipping the records mapped to singleton buckets, this approach potentially suffers because of load imbalance: As we shall see in the next section, LSH naturally results in buckets of widely varying size. As a result, some bucket-vertices have to perform heavy IMM computations, which are of quadratic time complexity, whereas others are lightly loaded. We shall see that even with careful distribution of bucket-vertices to processors, this still results in significant load-imbalance, or *skew*, causing inefficiency.

## 2.2 Record-centric Approach in Pregel (RCP)

The motivation for record-centric parallelization (RCP) is to overcome the problems of skew by distributing the IMM computations for records mapped to the same bucket *back* to the record-vertices themselves. The load for large IMM jobs at bucket-vertices

is thus further parallelized. Record-vertices end up with work assigned to them from at most  $b$  buckets where they are mapped to; as a result the computations are better balanced even when record-vertices are randomly distributed across processors. Of course, the cost for such further re-distribution of load is increased communication cost. (A detailed cost analysis of both approaches is presented in the next section.)

RCP comprises of seven supersteps using the graph-parallel Pregel paradigm as explained below and summarised in Figure 3. Note however, that this sequence of seven supersteps will run iteratively many times until all vertices halt and no further messages are generated. As before, we assume that the initial LSH phase is performed using MR to instantiate a graph comprising of *all* records but only non-singleton buckets.

**SS1:** Every bucket-vertex sends messages to the record-vertices that are hashed to it in order to *schedule* their comparisons as per the IMM algorithm: For each pair of record-ids  $\{r_i, r_j\}$  from the set of record-ids hashed to the bucket-vertex, a message is sent to one of the two record-vertices: Say  $\{r_j\}$  is sent to  $r_i$  if  $i < j$ , otherwise the message  $\{r_i\}$  is sent to  $r_j$ . After sending all such messages, the bucket-vertex votes to halt. Also, a record-vertex does nothing in this superstep, and votes to halt. (Unless it is a lone record that is not present in any non-singleton bucket, it will get woken up via messages in the next step; in general, vertices perform their work in a superstep and vote to halt, only to be woken up via messages later.)

If the outgoing edge-list of a bucket-vertex consists of  $k$  records  $\{r_1, r_2, \dots, r_k\}$ , then  $r_1$  will be sent  $\{r_2, \dots, r_k\}$  messages,  $r_2$  will be sent  $\{r_3, \dots, r_k\}$  messages, and so on. This message-sending scheme ensures that if a pair of records co-exists in more than one bucket, then the same record-vertex (the record-vertex with lower id) will receive the messages from all such buckets. For the graph in Figure 1, bucket-vertex  $b_1$  has  $\{r_1, r_2, r_4\}$  in its neighborhood and it sends messages to  $r_1$ :  $\{r_2, r_4\}$ , and to  $r_2$ :  $\{r_4\}$ . Similarly,  $b_2$  sends a message  $r_3$  to  $r_2$ .

**SS2:** Before this superstep, every record-vertex is inactive, and gets activated when it receives IDs of record-vertices that it needs to be compared with. A record-vertex now sends its *value* (containing the full record) to the record-vertices whose IDs were received in messages.

Note that if two record-ids  $r_i$  and  $r_j$  (with  $i < j$ ) co-occur in edge-lists of  $k$  bucket-vertices,  $r_i$  will receive  $k$  messages (one from each of the  $k$  bucket-vertices), all containing the record-id  $r_j$ . The record-vertex  $r_i$  will send its value to  $r_j$  only once. For the graph in Figure 1, record-vertex  $r_1$  sends its value to  $r_2$  and  $r_4$  based on the messages received from  $b_1$ . Similarly,  $r_2$  sends its value to  $r_3$  and  $r_4$ , based on the messages received from bucket-vertices  $b_2$  and  $b_1$ , respectively.

**SS3:** Now actual comparisons between records takes place via the match function. A record-vertex  $r$  receives messages containing the values of the record-vertices it has to be compared with. Note that the message sending scheme in SS2 ensures that even if a pair of record-ids co-occur in more than one bucket they get compared only once.

If the value of the record-vertex  $r$  matches the value of an incoming message  $r'$ , a message  $\{r, r'\}$  containing the IDs of the two matched vertices is sent to  $r$  and  $r'$ . For the graph in Figure 1, in SS3, record-vertex  $r_2$  receives the value of  $r_1$ ,  $r_3$  receives the value of  $r_2$ , and  $r_4$  receives the values of  $r_1$  and  $r_2$ . At vertex  $r_2$ , values of  $r_1$  and  $r_2$  are compared and are found to be matching, and the message  $\{r_1, r_2\}$  (containing only the IDs of the two vertices) is sent by  $r_2$  to both  $r_1$  and  $r_2$ . Similarly the message  $\{r_2, r_3\}$  is generated at  $r_3$ , and is sent to both  $r_2$  and  $r_3$ . The messages  $\{r_1,$

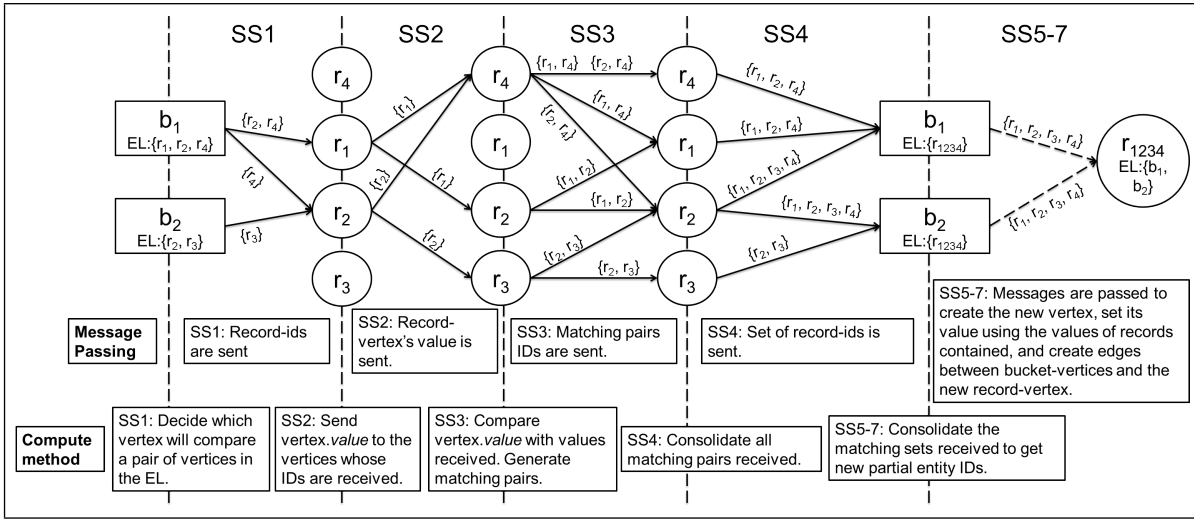


Figure 3: RCP: Supersteps SS1 to SS7 for the example in Figure 1. Here, EL stands for outgoing edge-list.

$r_4$  and  $\{r_2, r_4\}$  are generated at record-vertex  $r_4$ . The message  $\{r_1, r_4\}$  is sent to  $r_1$  and  $r_4$ , and the message  $\{r_2, r_4\}$  is sent to  $r_2$  and  $r_4$ .

**SS4:** If a record  $r_i$  matched with any record  $r_j$  in the previous superstep, it will receive a message  $\{r_i, r_j\}$ . If  $r_i$  matched  $m$  records in the previous superstep, it receives  $m$  messages in this superstep: one from each of the  $m$  matches. Since  $r_i$  matches all the  $m$  records, all the  $m + 1$  records (including  $r_i$ ) belong to the same entity. The record-vertex consolidates all the pairs of IDs received as messages into a set containing the  $m + 1$  IDs all belonging to the same entity. This set of IDs is sent to all the bucket-vertices in the outgoing edge-list of the record-vertex  $r_i$ .

For the graph in Figure 1,  $r_1$  receives  $\{r_1, r_2\}$  and  $\{r_1, r_4\}$ , and consolidates them to create a message  $\{r_1, r_2, r_4\}$  which is sent to  $b_1$ . Record-vertex  $r_2$  consolidates  $\{r_1, r_2\}$ ,  $\{r_2, r_3\}$  and  $\{r_2, r_4\}$  to get  $\{r_1, r_2, r_3, r_4\}$  which is sent to both  $b_1$  and  $b_2$ . Similarly,  $r_3$  sends  $\{r_2, r_3\}$  to  $b_2$ , and  $r_4$  sends  $\{r_1, r_2, r_4\}$  to  $b_1$ .

**SS5:** Similar to the previous superstep, where a record-vertex consolidates the matching information about itself, a bucket-vertex consolidates all the sets received as messages and creates new record-ids as follows: If any two sets  $s_i$  and  $s_j$  received by a bucket-vertex have an element (a record-id) in common, i.e.  $s_i \cap s_j \neq \emptyset$ , all the record-ids in the two sets belong to the same entity. A new set  $s_{ij} = s_i \cup s_j$  is created, and the sets  $s_i$  and  $s_j$  are deleted. This is done iteratively till all the sets remaining are disjoint.

Now new record-vertex needs to be created for each of the disjoint sets, so the bucket-vertex sends a graph-mutation request to create these new vertices, which we shall call *partial-entity-vertices*. Mutation requests are also sent to create bi-directional edges between the partial-entity-vertex and the bucket-vertex. The IDs of these vertices are based on the final consolidated sets  $s_{ij}$ , so if the same partial-entity-vertex is created by multiple buckets, this does not lead to duplicate vertices. The bucket-vertices also send a message to each record-vertex they are connected with, informing the record-vertices about their corresponding partial-entity-id.

For the graph in Figure 1, bucket-vertex  $b_1$  receives 3 messages  $\{r_1, r_2, r_4\}$ ,  $\{r_1, r_2, r_3, r_4\}$  and  $\{r_1, r_2, r_4\}$  which are consolidated to get a set  $\{r_1, r_2, r_3, r_4\}$ . Bucket-vertex  $b_1$  sends a message to create a partial-entity-vertex with id  $r_{1234}$ . Similarly, bucket-vertex  $b_2$  receives 2 messages  $\{r_1, r_2, r_3, r_4\}$  and  $\{r_2, r_3\}$  which

are consolidated to get  $\{r_1, r_2, r_3, r_4\}$ , and a message is sent by  $b_2$  to create a new vertex  $r_{1234}$ . Here, both the bucket-vertices  $b_1$  and  $b_2$  ask for the creation of the same vertex since both have computed the same consolidated set:  $\{r_1, r_2, r_3, r_4\}$ . Also,  $b_1$  sends a message to create a bi-directional edge between  $b_1$  and  $r_{1234}$ . Similarly,  $b_2$  sends a message to create a bi-directional edge between  $b_2$  and  $r_{1234}$ . Also,  $b_1$  and  $b_2$  send a message containing the id  $r_{1234}$  to all their record-vertices.

**SS6:** New partial-entity-vertices get created before start of this superstep. A record-vertex  $r$  receives messages containing the id of a new partial-entity-vertex  $r'$ . The record-vertex  $r$  sends its value (i.e., record content) and its outgoing edge-list as a message of the form  $\{v_i, e_i\}$  (where  $v_i$  is the value of vertex  $r_i$  and  $e_i$  its outgoing edge-list) to  $r'$ . For the graph in Figure 1, the record-vertices  $r_1, r_2, r_3$  and  $r_4$  send  $\{v_1, e_1\}$ ,  $\{v_2, e_2\}$ ,  $\{v_3, e_3\}$ , and  $\{v_4, e_4\}$  respectively to  $r_{1234}$ .

**SS7:** In this superstep, a partial-entity-vertex  $r$  receives messages of the form  $\{v_i, e_i\}$ . The value (i.e., content) of  $r$  is initialised by merging all the record content values  $v_i$ s received. The vertex  $r$  also sends messages to create outgoing edges with every bucket-vertex whose ID is present in the outgoing edge-lists  $e_i$ s received. For every bucket-vertex  $b_i$ , to which a partial-entity-vertex  $p_i$  is added,  $p_i$  needs to be compared with the other records and partial-entities in  $b_i$ 's edge-list. So a message is also sent to these bucket-vertices to activate them before the beginning of next iteration. The partial-entity-vertices are treated like record-vertices for next iteration of supersteps SS1 to SS7.

Finally, each partial-entity-vertex  $r$  sends graph-mutation requests to delete every vertex  $r_i$  (corresponding to  $\{v_i, e_i\}$ ) that led to  $r$ 's creation (also for deleting all incoming and outgoing edges of  $r_i$ ). For the graph in Figure 1,  $r_{1234}$  receives the values of record-vertices  $r_1, r_2, r_3$ , and  $r_4$  as messages, and uses them to update its value. Also, messages are sent to create bi-directional edges between  $r_{1234}$  and  $b_1$ , and  $r_{1234}$  and  $b_2$ . Messages are also sent to delete  $r_1, r_2, r_3$ , and  $r_4$ , and activate  $b_1$  and  $b_2$ .

*Iterations in RCP:* In the first iteration of the algorithm, i.e., at the beginning of SS1, all bucket-vertices are active. In the beginning (i.e., SS1) of each subsequent iteration, only those bucket-vertices are active that receive messages from SS7 of the previous iteration. Iterations continue until no more messages are generated.

A bucket-vertex can have both old and new record-ids in its out-

going edge-list at the end of any iteration of supersteps SS1 to SS7. Record-id pairs for a bucket-vertex which have already been compared need not be compared again. To avoid such comparisons, a set  $P$  is maintained for each bucket-vertex which contains the pairs which have already been compared in previous iterations. For example, suppose a bucket  $b$  has 4 records  $\{r_1, r_2, r_3, r_4\}$  in its outgoing edge-list. After the first iteration of supersteps SS1 to SS7, suppose  $r_1$  and  $r_2$  get merged to form a new record  $r_{12}$ . Then the outgoing edge-list of  $b$  will be  $\{r_{12}, r_3, r_4\}$ , and the set  $P = \{\{r_1, r_2\}, \{r_1, r_3\}, \{r_1, r_4\}, \{r_2, r_4\}, \{r_3, r_4\}\}$ . So, in the next iteration only the following record pairs need to be compared:  $\{\{r_{12}, r_3\}, \{r_{12}, r_4\}\}$ . For the graph in Figure 1, buckets  $b_1$  and  $b_2$  will have only one record-id  $\{r_{1234}\}$  in their respective outgoing edge-lists. So, no further comparisons are done and no further messages are sent, terminating the algorithm.

### 3. COMPARISON OF BCP AND RCP

Some of the notations used for the analysis in this section have been presented in Table 1. To compare the parallel execution times of the two approaches presented in Section 2 we first note that total execution time comprises of (a) total computation cost ( $T_n$ ) and (b) total communication cost ( $T_c$ ) [10]. In an ideal situation, the parallel computation time on  $p$  processors should be  $T_n(p) = T_n/p$ . Further, assuming a fully interconnected inter-processor network, communication between two disjoint pairs of processors can take place in parallel; if all communication is parallel in this manner then the parallel communication time on  $p$  processors should be  $T_c(p) = T_c/p$ .

Therefore, the ideal parallel execution time  $T(p)$  should be the sum of  $T_n(p)$  and  $T_c(p)$ . However, this is not the case in practice. First, computation load is often unevenly distributed across processors; say the most heavily loaded processor having  $s_1$  times more work than the average processor, which we refer to as *computation skew*. For example, in case of LSH, commonly used words lead to high textual similarity between a lot of records, and all these records may get hashed to the same bucket. So some buckets end up with a very large number of records. Similarly, communication need not be evenly balanced either: For example, a source processor sending a message to all other processors takes  $p - 1$  steps in spite of the fact that each receiving processor gets only a single message; the source processor becomes the bottleneck. *Communication skew*  $s_2$  is similarly defined as the ratio between the heaviest and average communication time expended by processors. Taking skew into account, we note that the parallel execution time on  $p$  processors satisfies:

$$T(p) = (s_1 T_n + s_2 T_c)/p \quad (1)$$

In the context of entity resolution via BCP and RCP, messages between processors are of two types: a) messages containing only vertex-ids, b) messages containing an entire record. In our analysis, we assume that latter dominates and so in our analysis we ignore messages carrying only ids. Also, we ignore the cost of sending the values of record-vertices to the entity-vertices in the final merging process (when values of entity-vertices are updated) in both BCP and RCP, assuming it to be of the same order in both cases. Further, assuming uniform distribution of vertices across processors, if a message has to be sent from one vertex to another, the probability that the message will be sent to another processor is  $(p - 1)/p$ . For large enough  $p$  we can assume that the message is almost always sent to another processor.

We assume that computation cost is directly proportional to the number of pairs of records to be compared. Therefore, total computation cost  $T_n = w.\alpha$ , assuming  $\alpha$  pairs of records were compared,

Parameter	Description
$n$	Total number of records
$R$	Cost to send a record to another processor
$w$	Cost of one comparison using <i>Match</i> function
$b$	Number of buckets each record is hashed to
$t$	Total number of pairs across all buckets
$u$	Number of unique pairs of records
$f$	Replication factor $(t - u)/t$
$N$	Maximum number of records in a bucket
$d$	Maximum number of records in an entity
$p$	Total number of processors

Table 1: Parameters used

and  $w$  is the time taken for comparison of a pair of records. Similarly, communication cost is assumed to be directly proportional to the number of record-carrying messages exchanged between processors. Therefore,  $T_c = R.\beta$ , assuming  $\beta$  messages were exchanged between various processors, and  $R$  is the time taken for communication of one record between a pair of processors. Using (1) the parallel execution cost with  $p$  processors is as in (2) below, using which we proceed to compare the BCP and RCP approaches.

$$T(p) = (s_1.w.\alpha + s_2.R.\beta)/p \quad (2)$$

#### 3.1 BCP: Parallel Analysis

The BCP approach distributes LSH buckets across processors and runs IMM within each bucket. If a bucket has  $x$  records, in the best case, IMM performs  $x - 1$  comparisons: This happens when all the records belong to the same entity and each successive match operation succeeds. For example, if there are 4 records  $d_1, d_2, d_3$ , and  $d_4$  in a bucket, then there will be a minimum of 3 match operations on  $(d_1, d_2)$ ,  $(d_{12}, d_3)$ , and  $(d_{123}, d_4)$ . However, according to [4] the worst case computation cost of IMM on a block with  $x$  records and  $x'$  entities is  $(x - 1)^2 - (x' - 1)(x' - 2)/2$ . This will be maximum when  $x' = 1$ . Therefore, in the worst case a bucket with  $x$  records will execute  $(x - 1)^2$  comparisons.

For the average-case analysis, we assume  $\binom{x}{2}$  comparisons in a bucket of size  $x$ , i.e., the number of pairs in the block. So, the total number of comparisons is just  $t$ , the number of pairs of records across all blocks, i.e.,  $\alpha_{BCP} = t$ .

The maximum total communication is  $n.b$  since every record is sent to at most  $b$  buckets. However, a record need not be sent to a singleton bucket containing only itself. If  $k_1$  is the expected fraction of non-singleton buckets per record, with  $0 < k_1 \leq 1$ , the total communication cost for BCP is  $\beta_{BCP} = k_1.n.b$ . Using the above in (2) the total computation and communication costs for BCP are summarized in Table 2.

#### 3.2 RCP: Parallel Analysis

In RCP a bucket with  $x$  records schedules  $\binom{x}{2}$  comparisons that are executed at processors holding records, rather than at the processors holding the bucket itself. Further, unlike BCP, RCP involves multiple iterations: Nevertheless, RCP will involve at most  $d - 1$  iterations, if  $d$  is the maximum number of records an entity can have.

For an average-case analysis we assume that if there are  $x_i$  records in a bucket in the  $i$ th iteration, then  $x_{i+1} = \lceil x_i/2 \rceil$ . Consequently, if there are  $t_i$  comparisons in the  $i$ th iteration,  $t_{i+1} \approx t_i/4$ . Therefore, the total number of comparisons across  $d - 1$  iterations will be  $[t + t/4 + t/4^2 + \dots + t/4^{(d-2)}] = \frac{4}{3}t(1 - 4^{-(d-1)})$ .

Further, in RCP, a pair of records is compared only once (see SS3 in Section 2.2) even though it may occur in multiple buckets,



Approach	pT(p)
BCP	$s_1.w.t + s_2.k_1.R.n.b$
RCP	$s'_1.(1-f).k_2.w.t + s'_2.(1-f).k_2.R.t$

Table 2: Average-case parallel execution times

with replication factor  $f$  (see Table 1). Therefore, the total number of comparisons (in first iteration) is  $(1-f)t$  rather than  $t$ . As a result,  $\alpha_{RCP} = (1-f).k_2.t$ , where  $k_2 = \frac{4}{3}(1-4^{-(d-1)})$ .

Further, for each pair of records that need to be compared, one of the records in the pair is sent to the other’s processor. Therefore, the number of messages is same as the number of comparisons, so  $\beta_{RCP} = (1-f).k_2.t$ . Using the above in (2) the parallel execution time for RCP is also summarized in Table 2.

### 3.3 Skew Analysis and Inferences

Computation skew  $s_1$  and communication skew  $s_2$  depend on the distribution of records to the buckets. Let the number of buckets having  $x$  records be  $f(x)$ . We analyze the case when  $f(x)$  follows a power-law distribution (which is what we observed empirically as shown in the next section), i.e.,  $f(x) = c/x^r$ , where  $c$  and  $r$  are positive constants.

Further,  $\binom{x}{2} \geq x$ , for  $x \geq 3$ ; and  $\binom{x}{2}.f(x) \geq x.f(x)$ , for  $f(x) \geq 0$ . Therefore, the total number of pairs  $t = \sum_{x=2}^N \binom{x}{2}.f(x)$  is much more than  $nb = \sum_{x=1}^N x.f(x)$ , the total number of records in all buckets. As a result, we can assume that  $t \gg nb$ . So when the computation and communication skews for BCP and RCP are comparable, i.e.,  $s_1 = s'_1$ , and  $s_2 = s'_2$ , and the replication factor  $f$  is small, we observe from Table 2 that the computation cost of both approaches is of the same order, whereas the communication cost of BCP ( $s_2.k_1.R.n.b$ ) is clearly lower than that of RCP ( $s'_2.k_2.R.t$ ). As a result, it appears that BCP will perform much better than RCP.

However, in most of the situations, computation skew  $s_1$  of BCP, will be so high that it will become the dominant component of the parallel execution cost. To explain this, let us compare the effect of a large bucket in the two approaches. In BCP, a large bucket with size  $x$ , will result in  $\binom{x}{2}$  computations at the processor responsible for this bucket; this processor can become slow and can increase the parallel execution cost. However, in RCP, a large bucket results in less computation skew, since the  $\binom{x}{2}$  comparisons are distributed to  $x-1$  record-vertices that, on the average, will reside on different processors for large enough  $p$ . Thus the computation load gets distributed.

Further, even though the large bucket causes  $\binom{x}{2}$  messages to schedule comparisons across processors in case of RCP, these messages only contain the IDs of the record-vertices and as we have assumed earlier, this cost is small enough to be ignored. With this assumption, the number of record-bearing communications arising from a bucket of size  $x$  are those where the records send themselves to each other, i.e., at most  $x-1$  messages for one of the records in the bucket and  $x/2$  on the average. This is comparable to the communications required in BCP where  $x$  records need to be received at the bucket of size  $x$ . Therefore, the communication skews in both BCP and RCP are comparable whereas the computation skew caused by the largest bucket in case of RCP is likely to be less than the computation skew caused in case of BCP. In the next section we demonstrate this empirically with realistic data.

### 3.4 Empirical Comparison of BCP and RCP

We generated synthetic data of 1.2 million ‘residents’ ( $n = 1.2M$ ) as follows: We began with 100,000 seed records, where a

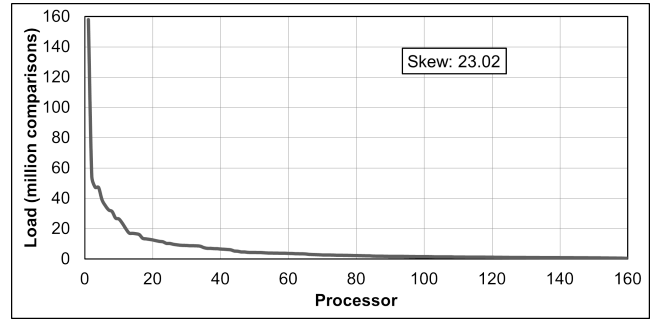


Figure 4: Average-case Computation Load for BCP

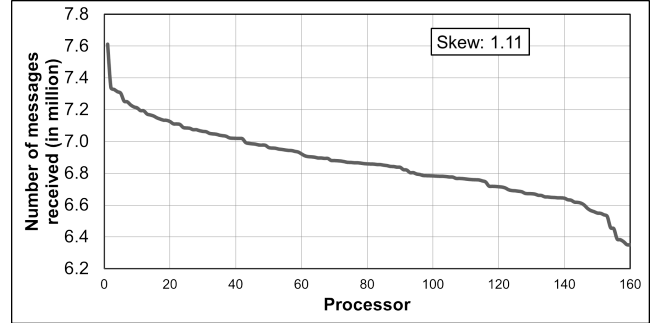


Figure 5: Average-Case Communication Load for BCP

seed record has all the information about an entity. For each such entity, a maximum of 5 records ( $d = 5$ ) are created corresponding to the following 5 domains: ‘Election Card’, ‘Income-Tax Card’, ‘Driving-Licence’, ‘Bank Account Details’, and ‘Phone Connections’. An entity can have a maximum of one record per domain. We control the creation of a record of a particular type for an entity using a random-number generator. To create a record from the seed record, values of some of the attributes of the person are omitted. For example, e-mail id of a person may be present in 2 of his records and absent in others. The values of the attributes of an entity across records are varied by using different methods. For example, the address of the different records for the same entity need not be same. Variation in values of attributes for different records for an entity are inserted by introducing typographical errors, swapping the first and last name, or omitting the middle name. To add further ambiguity after creation of the records for an entity, additional records which share considerable textual similarity with each other are generated for related entities, such as parent, child, spouse, neighbour, etc.

We applied 90 hash functions on each record and used  $a = 3$  and  $b = 30$  in the LSH formulation (refer Section 2.1.1). We get  $\approx 17.29M$  buckets, 59.34% of the buckets (10.26M) were singletons, 99.89% (17.27M) of the buckets had  $\leq 30$  records, 0.02% (696) buckets had  $> 100$  records, and 0.00092% (163) buckets had  $> 1,000$  records. (With this generated data, the value of  $r$  in power law distribution of records to buckets, i.e.,  $f(x)$  as defined earlier in Section 3.3 was found to be  $\approx 2.8$ .) The distribution of  $f(x)$  vs  $x$  is shown in Figure 7. The 163 (0.00092%) large buckets are the bottlenecks in terms of the execution-time taken by the processors which have them. The size of the largest and second largest buckets are 17,668 and 9,662, respectively.

To estimate the values of skew factors,  $s_1$ ,  $s_2$ ,  $s'_1$ , and  $s'_2$ , we assume the number of processors  $p = 160$  (corresponding to our

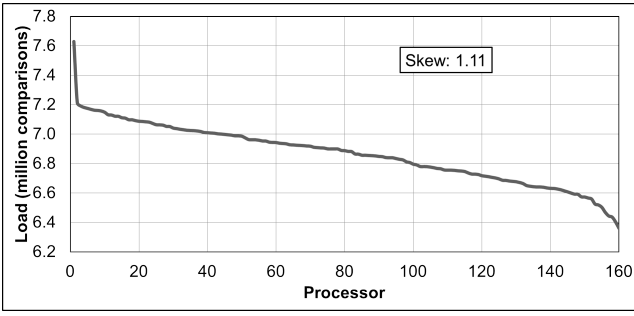


Figure 6: Average-Case Computation Load for RCP

Approach	Comparisons (millions)			Communications (millions)		
	Max.	Avg.	$s_1$	Max.	Avg.	$s_2$
BCP	158,039	6,864	23.02	0.171	0.154	1.11
RCP	7,629	6,864	1.11	7,612	6,864	1.11

Table 3: Skew factors for BCP and RCP (average-case)

physical cluster of five 4 CPU nodes with 4 cores and two virtual processors per core). The records and buckets (1.2M+17.29M vertices) were randomly distributed to these 160 processors. Assuming  $w = R = 1$ , the computation and communication loads per processor are shown in Figures 4, 5, and 6.

The skew factors as estimated from the above distributions are shown in the Table 3: It can be observed that the computation skew  $s_2$  in BCP is 23.02, whereas the communication skew of BCP  $s_1$  and the computation and communication skews  $s'_1$  and  $s'_2$  are almost the same ( $= 1.11$ ). Based on the formulations in Table 2, it is clear that assuming  $w = R = 1$ , replication factor  $f = 0$ ,  $d = 5$  so that  $k_2 = \frac{4}{3}(1 - 4^{-(d-1)}) \approx \frac{4}{3}$ ,  $k_1 = 1$ , the parallel execution-time of BCP,

$$\begin{aligned}
 p.T^b(p) &= s_1.w.t + s_2.k_1.R.n.b \\
 &= 23.02 \times t + 1.11 \times n.b \\
 &\geq 23.02 \times t
 \end{aligned}$$

will be much larger than that of RCP,

$$\begin{aligned}
 p.T^r(p) &= s'_1.(1-f).k_2.w.t + s'_2.(1-f).k_2.R.t \\
 &= 2 \times 1.11 \times \frac{4}{3}t \\
 &= 2.96 \times t
 \end{aligned}$$

since  $s_1$  is much higher than both  $s'_1$  and  $s'_2$ .

It may appear surprising that computation skew in BCP is so much higher than communication skew; after all the number of pairs at a bucket (which determines computation cost) is at most

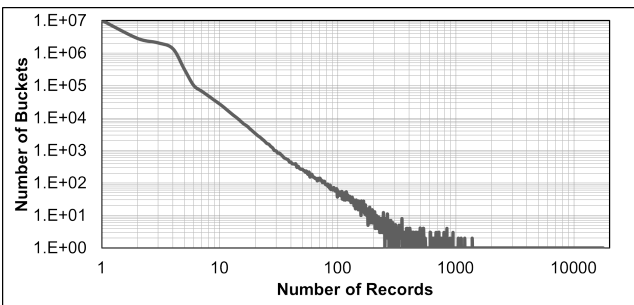


Figure 7: Number of Buckets ( $f(x)$ ) vs. Number of Records ( $x$ )

the square of the bucket's size (which determines communication cost). Computation skew, on the other hand, is much worse than the square of communication skew.

However, consider the following example taken from our synthetic data: Suppose we have 100 processors, and 150 buckets have size  $\geq 1000$ . Assuming uniform distribution, each processor gets 0.16M (16M total buckets / 100 processors). Suppose the sizes of the largest buckets  $b_1$  and  $b_2$  at processors  $P_1$  and  $P_2$  be 18,000 and 300 respectively. There are 0.16M - 1 other buckets each at  $P_1$  and  $P_2$ . Assuming uniform distribution of load on these two processors based on the 0.16M - 1 buckets (not considering the two buckets  $b_1$  and  $b_2$ ), the ratio of execution-times on  $P_1$  and  $P_2$  would be close to 1. Suppose the execution-time of these 0.16M - 1 buckets (all buckets except  $b_1$  in case of  $P_1$  and  $b_2$  in case of  $P_2$ ) is  $T$  on both the processors. Then the execution-time  $T_1$  on processor  $P_1$  is  $T + \binom{18,000}{2} = T + 161,991,000$ , and  $T_2$  on processor  $P_2$  is  $T + \binom{300}{2} = T + 44,850$ . To compute  $T$ , suppose the 60% of the singleton buckets get uniformly distributed. So, 96k (60% of 0.16M) of the buckets at  $P_1$  and  $P_2$  each have no contribution to  $T$ . Suppose the remaining 64k buckets get 30 records each (which gives an approximate upper bound on  $T$  assuming 99.9% of the buckets have less than 30 records). Then,  $T = 64,000 \times \binom{30}{2}$ , i.e., 27,840,000, so that  $T_1 = 189,831,000$  and  $T_2 = 27,884,850$ . So the computation skew  $\frac{T_1}{T_2} \approx 6.8$ . Empirically, even with uniform distribution of buckets across processors, we still find  $s_1^b \approx 20$ .

Thus, even if the distribution of 150 large-sized buckets is uniform across the 100 processors, still any processor getting two large-sized buckets results in significant computation skew for BCP. On the other hand, since RCP distributes computations by records, with the number of computations at a record directly proportional to the sum of its bucket sizes (rather than the square of bucket size as in BCP), the resulting computation skew is relatively small.

## 4. RELATED WORK

Parallel entity resolution (ER) using three distributed computing paradigms has been described in [23]: distributed key-value stores, MR, and bulk synchronous parallelism. Their strategy is to perform pair-wise comparisons followed by connected components. The bulk synchronous parallel (BSP) algorithm in [23] is related to the graph-parallel model. However, they do not use any acceleration strategy (such as LSH) to group candidate pairs; neither do they deal with the issue of load-imbalance (skew).

Other parallel implementations for entity resolution have been proposed in [3, 11, 7, 13], but none of these address the issue of skew. The D-Swoosh family of algorithms in [3] implements R-Swoosh based IMM [4] by distributing the task of comparing records to multiple processors by using *scope* and *responsibility* functions, where the *scope* function decides the processors where a record will be sent to, and the *responsibility* function decides which processor will compare a given pair of records. In one particular domain-dependent strategy called *Groups*, records are assigned to groups, and two records are compared only when they belong to the same group. This is similar to the idea of buckets in the context of BCP algorithm. However, in *Groups*-based D-Swoosh, each group is assigned to one processor, whereas in BCP, multiple buckets end up being assigned to the same processor.

The Dedoop tool described in [14] borrowing ideas from another work by the same authors in [15], is a MR-based entity resolution tool that includes different blocking techniques, provides strategies to achieve balanced workloads, and also provides redundant-free

comparisons when multiple blocking keys are used. Load balancing in Dedoop is achieved through an additional MR step before the actual matching job to analyze the input data and create a block distribution matrix (BDM). This BDM is then used by, for example, the BlockSplit strategy to split the match tasks for large-sized blocks into smaller match tasks for sub-blocks, ensuring all comparisons for the large-sized block gets done. Dedoop considers only pair-wise comparisons and does not perform IMM for which their load-balancing strategy may not work. Similar to our RCP approach, where a pair of records co-occurring in multiple buckets is compared only once, Dedoop also has provision for avoiding redundant comparisons when multiple blocking keys are used by comparing record-pairs only for their smallest common blocking key.

The stragglers problem in the context of MR, in general, has been discussed in [17, 21, 16]. The basic idea to avoid load imbalance, due to large number of values for a particular reduce-key, is to somehow split the keys with large loads into multiple sub-keys that can then be assigned to different reducers. Using such solutions in the context of Iterative Match-Merge for the values (records) at a reduce-key is not straight-forward and can be a direction for future research.

In [17], it is shown that when the distribution  $f(x)$  of the number of reduce-keys receiving  $x$  values follows a Zipf distribution, i.e.  $f(x) \propto \frac{1}{x^\alpha}$ , and each reduce task performs  $O(x)$  work, then the maximum speedup is around 14. This suggests that the situation will be worse in our case when the amount of work done for the task with  $x$  elements is  $O(x^2)$ , thereby justifying our RCP approach further.

Various grouping techniques such as sorted-neighborhood indexing and Q-gram-based indexing have been proposed to reduce the set of candidate pairs, as surveyed in [6]. Locality Sensitive Hashing (LSH) for entity resolution has been discussed in [12].

## 5. CONCLUSIONS AND LEARNINGS

We set out to develop a parallel implementation of entity resolution so as to handle with cases involving billions of records and hundreds of millions of entities. The bucket-parallel approach, which is natural using MR, results in significant skew. The record-parallel approach emerged as a natural alternative and turned out to have better load-balancing properties especially in the presence of severe skew, which arises naturally in hashing where some buckets corresponding, say, to very common names, end up with a large number of records.

Many problems involving evaluating pair-wise similarity of large collections of objects can be efficiently accelerated using probabilistic hashing techniques such as LSH, in much the same manner as we have accelerated IMM-based entity resolution. Clustering using canopies, similarity joins, feature-based object search as well as duplicate-detection in object databases (such as for biometric applications) are some such examples. In all such cases parallel implementation can be done bucket-wise or record-wise, and the advantages of the record-parallel approach can be derived whenever hashing is expected to result in skew due to some features being much more heavily shared across objects than others.

Last but not least, note that even in the BCP approach, we avoid sending records to singleton buckets; i.e., we first form buckets using just record-ids and then send the more voluminous actual records only to non-singleton buckets. Since we find that over 60% buckets are singletons in our sample data, this optimisation is fruitful even though it costs an additional communication step. Upon reflection we realised that other scenarios present a similar opportunity for optimisation: Consider a multi-way join implemented in

MR, but where some attributes are large objects such as images or videos. Even for normal joins (on say, object-id, i.e., not similarity joins), traditional MR implementations would ship entire records to reducers, including those that never match with others and are therefore absent in the final result. In such cases, eliminating singletons via an initial phase is an important optimisation that applies both for MR as well as graph-based parallelization.

## 6. ACKNOWLEDGEMENT

The authors would like to thank Jeffrey D. Ullman for suggesting that while the RCP approach may have some advantages, the BCP approach would have lower communication costs; so a thorough analysis is in fact required.

## 7. REFERENCES

- [1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 459–468, 2006.
- [2] C. Avery. Giraph: Large-scale graph processing infrastructure on hadoop. *Proceedings of the Hadoop Summit. Santa Clara*, 2011.
- [3] O. Benjelloun, H. Garcia-Molina, H. Gong, H. Kawai, T. Larson, D. Menestrina, and S. Thavisomboon. D-swoosh: A family of algorithms for generic, distributed entity resolution. In *Distributed Computing Systems, 2007. ICDCS '07. 27th International Conference on*, pages 37–37, 2007.
- [4] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, 18(1):255–276, 2009.
- [5] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000.
- [6] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *Knowledge and Data Engineering, IEEE Transactions on*, 24(9):1537–1555, 2012.
- [7] P. Christen, T. Churches, and M. Hegland. Febrl – a parallel open source data linkage system. In H. Dai, R. Srikant, and C. Zhang, editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 638–647. Springer Berlin Heidelberg, 2004.
- [8] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web, WWW '07*, pages 271–280, New York, NY, USA, 2007. ACM.
- [9] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, Jan. 2008.
- [10] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker. *Solving Problems on Concurrent Processors. Vol. 1: General Techniques and Regular Problems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
- [11] H.-s. Kim and D. Lee. Parallel linkage. In *Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management, CIKM '07*, pages 283–292, New York, NY, USA, 2007. ACM.
- [12] H.-s. Kim and D. Lee. Harra: Fast iterative hashed record linkage for large-scale data collections. In *Proceedings of the*

- 13th International Conference on Extending Database Technology*, EDBT '10, pages 525–536, New York, NY, USA, 2010. ACM.
- [13] L. Kolb, H. Köpcke, A. Thor, and E. Rahm. Learning-based entity resolution with mapreduce. In *Proceedings of the Third International Workshop on Cloud Data Management*, CloudDB '11, pages 1–6, New York, NY, USA, 2011. ACM.
- [14] L. Kolb, A. Thor, and E. Rahm. Dedoop: efficient deduplication with hadoop. *Proceedings of the VLDB Endowment*, 5(12):1878–1881, 2012.
- [15] L. Kolb, A. Thor, and E. Rahm. Load balancing for mapreduce-based entity resolution. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 618–629, 2012.
- [16] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. A study of skew in mapreduce applications. *Open Cirrus Summit*, 2011.
- [17] J. Lin. The curse of zipf and limits to parallelization: A look at the stragglers problem in mapreduce. *Proceedings of 7th Workshop on Large-Scale Distributed Systems for Information Retrieval*, pages 57 – 62, 2009.
- [18] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, Apr. 2012.
- [19] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.
- [20] A. Rajaraman and J. Ullman. *Mining of Massive Datasets*. April 2010.
- [21] S. R. Ramakrishnan, G. Swart, and A. Urmanov. Balancing reducer skew in mapreduce workloads using progressive sampling. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 16:1–16:14, New York, NY, USA, 2012. ACM.
- [22] S. Salihoglu and J. Widom. Gps: A graph processing system. *Technical Report, Stanford University*, 2013.
- [23] C. Sidló, A. Garzó, A. Molnár, and A. Benczúr. Infrastructures and bounds for distributed entity resolution. In *9th International Workshop on Quality in Databases*, 2011.
- [24] T. White. *Hadoop: The Definitive Guide: The Definitive Guide*. O'Reilly Media, 2009.

# Modular Data Clustering - Algorithm Design beyond MapReduce

Martin Hahmann, Dirk Habich, Wolfgang Lehner

Technische Universität Dresden  
Department of Computer Science  
Database Technology Group  
01062 Dresden

{martin.hahmann; dirk.habich; wolfgang.lehner} @tu-dresden.de

## ABSTRACT

In the context of Big Data, flexible and adjustable data analytics become more and more important, whereas an efficient, scalable and fault-tolerant execution is required as well. To fulfill the flexibility as well as the execution requirements, the specification of the analysis methods have to be in an appropriate and easy adjustable manner. The MapReduce approach has demonstrated that such flexible specification as well as scalable execution is possible and applicable. However, the MapReduce programming model is too generic and complicates the specification from a data analysis point of view. Therefore, we propose a novel programming approach using well-defined modular building blocks for a specific and highly utilized data analysis domain named data clustering in this paper. Our approach offers many advantages: (i) a unified and specific instruction set for data clustering which eases understanding and algorithm adaptation in an abstract way, and (ii) enables an efficient and scalable execution of all data clustering algorithms based on an efficient mapping of the unified instruction set to a specific target environment is possible.

## 1. INTRODUCTION

In order to efficiently process and analyze massive data, highly scalable parallel data processing platforms have been developed [16]. In this area, MapReduce [7] is a well-established programming and execution framework. A MapReduce cluster is able to scale to thousands of commodity computer nodes in a fault-tolerant manner. Furthermore, the programmers can parallelize their applications in an easy way by implementing map and reduce functions to transform and aggregate data, whereas the underlying data structure consists of *(key, value)* pairs. As shown in different application domains, many algorithms fit perfectly into the MapReduce model, such as word counting in information retrieval or equi-join queries in databases.

Generally, the success of MapReduce is based on the simple and flexible programming model with the ability to execute the applica-

tion in a highly parallel and fault-tolerant fashion [7, 16]. Nevertheless, the MapReduce model has several shortcomings. For example, one drawback of MapReduce is the missing support for iterative computations. However, many data analysis techniques require those iterative computations, therefore Bu et al. [4] have proposed an extension to MapReduce. A second drawback is that MapReduce itself does not support any high-level language like SQL in database systems. Users always have to code their operations in map and reduce functions, whereas they have to consider the *(key, value)* data structure. Mapping of individual data structures to *(key, value)* pairs is not always trivial [2]. To overcome this issue, a variety of projects aim at providing higher-level interfaces. An example is Jaql<sup>1</sup> as a declarative query interface with rich data processing features such as transformation, filtering, join processing, grouping and aggregation. The Jaql scripts are automatically compiled and executed as MapReduce jobs.

However, the available higher-level interfaces focus on data transformation and processing tasks. To support deeper analytics as necessary for Big Data, Das et al. [6] integrate the statistical analysis system R<sup>2</sup> in the MapReduce system Hadoop. This integration is done on the language level by combining Jaql with R scripts as well as on the execution level using a R-Jaql bridge between R and Hadoop. The advantage of this approach is the efficient utilization of the rich functionalities of R for analytical tasks. From a usability point of view, this integration fits perfectly, because users can immediately start deep analytics on their data using standard functions. The disadvantage of this approach is availability of two different runtime infrastructures which have to interact to determine a result. Furthermore, this language approach of Jaql combined with R is not well suited for the specification of new analysis methods in a MapReduce style, so that these methods are finally scalable on a highly parallel platform.

To tackle the flexible specification or engineering of analytical methods for large scale analytics, we propose an alternative approach with regard to modular algorithm design inspired by MapReduce. We illustrate our approach using algorithms from data mining, in particular from the data clustering domain. Fundamentally, data clustering is a highly applicable analysis method that is used to reduce the amount of data or to gain understanding and acquire novel, previously unknown knowledge. The task of data clustering is to partition a set of objects into groups—so called clusters—in a way that similar objects are put in the same cluster, while dissimilar objects are located in different clusters. To determine such cluster-

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

<sup>1</sup><https://code.google.com/p/jaql/>

<sup>2</sup><http://www.r-project.org>

ing result, a large algorithmic landscape has been established. To get an idea of this landscape, in terms of size, we looked through the proceedings of the SIGKDD and ICDM data-mining conference from 2005 and 2012 and counted more than 120 papers introducing new algorithms or variants resp. optimization of existing techniques. This multitude of algorithms exists because it is impossible to design an algorithm that automatically produces optimal results for any data set or application, thus a lot of techniques are highly specialized and custom-made for specific scenarios or types of data sets.

### Our Contribution

In this paper, we present our modular language approach for the description and specification of clustering algorithms. Our design principles are (i) to establish a unified view on clustering algorithms using a compact set of instructions and (ii) to hide details of parallel execution on the programming level as successfully practiced by MapReduce. Both design principles enable users to focus only on the clustering analytical part, without considering the efficient execution on a scalable data processing platform. Generally, our approach is based on a mathematical formulation and utilizes a *matrix* concept for the unified representation of all data aspects. As a result, clustering specific operations are expressed as functions over matrices. From a language perspective, this eases understanding and engineering of clustering algorithms and allows their comparison. Furthermore, our modular approach offers an efficient way to adapt clustering algorithms. From an execution perspective, several different execution and optimization strategies are possible.

### Outline

In Section 2, we review essentials of data clustering by decomposing clustering algorithms into conceptual components. These conceptual components are concretized in Section 3 and 4 with a data model, building blocks in the form of matrix functions and control-flow structures. In Section 5, we demonstrate how clustering algorithms are transcribed using our approach. For this, we choose algorithms from different clustering classes to emphasize the wide-range applicability of our approach. The contributions of our approach and its potentials are described in detail in Section 6. The future development of our concepts is described in Section 7, before we review existing related work in Section 8. The paper closes with a short summary in Section 9.

## 2. ESSENTIALS OF DATA CLUSTERING

To reach our goal of a unified and specific instruction set for clustering algorithms, we first need to decompose the corresponding algorithms into their conceptual components. This decomposition concentrates only on the core clustering procedure and does not consider pre- and post-processing tasks like feature selection, data cleansing and so on. As starting point, we assume the a general definition of data clustering [12]: "*Data clustering is the partitioning of a set of points into groups—so called clusters—in a way that similar points are put in the same cluster, while dissimilar points are located in different clusters.*"

From this definition, we can derive certain fundamental tasks that have to be performed in order to generate a clustering. The first fundamental task an algorithm needs to fulfill, is to measure the similarity of points. This is a prerequisite for the second task, which is to explicitly choose the points that are similar and should be grouped together. The actual grouping of points, forms the third and final task that must be executed in order to create a clustering.

### 2.1 Basic Elements

The three identified tasks, which are observable in all clustering algorithms, are independent and have to be executed in sequence. As result of this abstract consideration, we define the *phases* of a clustering algorithm, that form the frame for our algorithm representation. This general frame needs to be fitted with additional building blocks to complete the description of an algorithm. In the following, we introduce each phase and investigate its basic elements, in order to find these blocks.

#### Evaluation Phase

During this first phase, the similarity between all points or between all points and some set of references is measured. For the determination of similarity between two objects, a dedicated function is necessary. In data clustering, there are two general approaches: (i) similarity functions and (ii) distance functions [12]. While the former express the degree of equality, the latter point out the amount of disagreement between objects. As both options are analogous, we assume that similarity is expressed through distances, without losing generality. Based on this, *distance measure* becomes the first basic element of the evaluation phase.

A distance measure takes at least two values as input and outputs one value. Obviously, in data clustering one input are the *points* which are to be clustered. The second input offers some kind of variability. On the one hand, there exist algorithms like DBSCAN [8] that calculate all point-to-point distances and thus use *points* also as second input. On the other hand, approaches like k-means [10] employ a special set of representatives/centroids as second input for distance computation. To combine both alternatives, we introduce the term *references* for the second input of the *distance measure*. The *references* can be (i) equal to *points*, (ii) a subset of *points* or (iii) a set of objects that are not part of *points* but share its feature-space. Following this, we add these two inputs to the set of basic elements for the evaluation phase.

The output of the distance measure consists of *distances* which is the next basic element. With this, we identified the four basic elements of the evaluation phase *points*, *references*, *distances*, and *distance measure* that allow a more specific definition of its task. In essence, during evaluation the distance measure is used to create a relation between points and references that represents their similarity and is explicitly expressed in the form of distances. It is important to mention that the distance itself is not the only result of evaluation, but the relation-triple *point-distance-reference*.

#### Selection Phase

In this phase, the points that are eligible to be grouped together are selected according to the algorithms specification. For this, the *point-distance-reference* triples generated by evaluation are taken as an input. Referring to our initial clustering definition, the goal is that only points which are similar should overcome the selection process in order to be clustered together. Therefore, it is necessary to define the constraints to acquire the status "similar" and to test all points whether they fulfill these or not. For this, we propose *filters*, which represent the basic element of this phase. Utilizing a set of *filters*, the selection phase tests each *point-distance-reference* triple coming from evaluation and only passes on those that comply.

#### Association Phase

During this phase, the previously chosen points are associated with a cluster. The input of the association phase is a set of *point-distance-reference* triples that passed the selection phase and have to be grouped together to create a clustering. This *clustering* forms the output of this phase and qualifies as basic element. Like the

Phase	Input	Processing	Output
Evaluation	Points, References	Distance Measure	Point-Distance-Reference Triples
Selection	Point-Distance-Reference Triples	Filters	Point-Distance-Reference Triples
Association	Point-Distance-Reference Triples	Association Function	Adjacencies (Point-Reference Tuples)
Optimization	-	-	-

**Figure 1: Overview Clustering Algorithm Phases and Basic Elements.**

similarities from evaluation, a clustering is made up of relations i.e. the affiliation between points and clusters. That means during the association phase *point-distance-reference* triples have to be transformed into *point-cluster* tuples. To perform this task, we define an *association function* as basic element.

For some algorithms, this is already enough to create a clustering e.g., the *association function* of k-means effectively takes a *point-distance-reference* triple, removes the distance and adopts the reference as cluster. But not all clustering techniques work in that way. For example, DBSCAN [8] first associates a core-object with its neighborhood—by creating *point-reference* tuples—before the actual clusters are formed on the basis of overlapping neighborhoods. Additionally, the direct creation of point-cluster tuples in DBSCAN is prevented by the fact that clusters are not known in advance. This issue necessitates a stopover between association function and clustering, which forms our next basic element: *adjacencies*. With it, we describe the association phase as follows: incoming *point-distance-reference* triples are transformed by the *association function* into *adjacencies* from which the clustering is derived. The transition from adjacencies to clustering is a part that can be done in a variety of ways, which is why we do not appoint basic elements for it on this conceptual level. Doing so would result in a substantial set of basic elements, that would be contradictory to our goal of finding only fundamental components. However, we solve this problem in a later section.

### Optimization Phase

This fourth phase originates from analyzing existing algorithms that often feature parameter adjustments and target function maximization leading to multiple iterations of the first three phases. As the name optimization implies, this phase is mainly concerned with improving the result generated by the preceding phases. Therefore, we assume that it is optional in contrast to the other three phases, that are mandatory for each clustering algorithm. The problem of variety we explained in the association phase, holds for this phase as a whole. As optimization can involve tasks like parameter adjustment, updates to points or references, iteration etc. the derivation of a minimal set of basic elements is not feasible at the moment. As stated before, we will solve this problem in a later section by moving to a different level of abstraction.

## 2.2 Summary

Figure 1 summarizes our conceptual decomposition of clustering algorithms. We identified three core phases which have to be executed and one additional phase for several optimizations. Furthermore, we defined the basic elements for each phase. In the following sections, we concretize our approach and iron out the flaws still existing at this point.

## 3. DATA MODEL

To realize our concept of a unified and clustering-specific instruction set, we have to define a data model for our presented input and output elements: *points*, *references*, *point-distance-reference triples (distances)*, *adjacencies* and *clustering* at first. In our approach, we propose to use *matrices* as a unified formal representation for all input and output basic elements.

### Points $P$ and References $R$

When it comes to the formal definition of a dataset for clustering, existing literature generally uses multi-dimensional vectors to represent the location of data points inside a feature-space. Following this procedure, we define our basic element *points* as a set  $P$  of  $f$ -dimensional vectors  $\vec{p} = \{p_0, \dots, p_f\}$  where  $(0 \leq j \leq f)$  and with  $n = |P|$ . This set can be easily converted into a matrix  $P$  by interpreting each vector  $\vec{p}_i$  as a row  $p_{i,*}$  of said matrix, thus giving  $P$  the dimensions of  $n$  rows and  $f$  columns. We stated earlier, that the set of *references* can either be a subset of  $P$  or just be located in the same feature space. This allows us to define it similar to  $P$ , as set of vectors  $R$ , containing  $\vec{r} = \{r_0, \dots, r_f\}$  where  $(0 \leq j \leq f)$  and  $k = |R|$ , which forms a matrix  $R^{k \times f}$ .

### Distances $D$ , Adjacencies $A$ , Clustering $C$

While  $P$  and  $R$  basically express the values of features per point, the remaining actors are instantiations of relations between objects e.g. *point-distance-reference* triples from evaluation or point-cluster tuples from clustering. For the formal description of these actors, matrices are especially convenient as the objects involved in the relation correspond to a row and column pair which addresses the matrix element holding the value of the actual relation. As an illustration, we define the basic element *distances* as a matrix  $D$  with  $n$  rows and  $k$  columns, where  $n = |P|$  and  $k = |R|$ . Each element  $d_{ij}$  of  $D$  relates to a point/row  $p_{i,*}$  of  $P$  and a reference  $r_{j,*}$  of  $R$ . Thus,  $d_{ij}$  contains the distance between  $p_{i,*}$  and  $r_{j,*}$ . Besides triples, this description can be translated smoothly to tuples like point-cluster from *clustering*. Such a tuple point-cluster expresses an existing relation in a binary fashion i.e. a relation between a point and a cluster only exists, if the corresponding tuple exists. This can be described with a binary matrix, where a value of 1 at position  $(i, j)$  indicates an existing relation between the objects referenced by  $i$  and  $j$ , while 0 states the opposite. Accordingly, we define *adjacencies* as binary matrix  $A$  having the same dimensions as  $D$ . To complete our description, we also define *clustering* as binary matrix  $C$  with  $n$  rows and a number of columns determined by the number of clusters found.

### Notation

Fundamentally, the notation for our reduced instruction set corresponds to a pseudocode notation. Matrices are denoted with a single capital letter e.g.  $D$  for the distances. Additional designation is done in the sub- and superscript of the letter. To distinguish different matrix versions we use the superscript:  $D^I$  and  $D^{II}$  are

versions of  $D$  after 1 resp. 2 function applications, while  $D^x$  and  $D^{x+1}$  designate the versions of  $D$  that are in effect for the current resp. next iteration of the algorithm. With the subscript, matrices can be described in more detail e.g.  $D_R$  denotes the distances between all references  $R$ .

## 4. BUILDING BLOCKS

So far, we defined the *matrix* as unified data model for our reduced instruction set in the previous section. Next, we have to find a fitting formal representation for the remaining basic elements like distance measure, filters and association function as well as a necessary set of control flow structures.

### 4.1 Functions

We describe the remaining basic elements as functions over matrices. Whereas, we use infix notation for elementary matrix functions: addition, subtraction, multiplication and entry wise multiplication, while prefix notation is used for all other functions.

---

$A \circ B \rightarrow C$	$\triangleright$ infix example: entry wise product
$function(A) \rightarrow A^I$	$\triangleright$ prefix example: single input function
$function(A, B) \rightarrow C^I$	$\triangleright$ prefix example: double input function

---

In the next step, we formally define the functions for the proposed basic elements distance measure, filters and association function.

#### Distance Function

We start with the distance measure *dist*, which takes a pair of rows  $(p_{i,*}, r_{j,*})$  from  $P$  and  $R$  and assigns a scalar value to it, that represents the distance between the corresponding objects. The abstract function *dist* can be defined as:

$$\begin{aligned} dist : \mathbb{M}^{n \times f} \times \mathbb{M}^{k \times f} &\rightarrow \mathbb{M}^{n \times k} \\ (P, R) &\mapsto D \\ d_{ij} &= f(p_{i,*}, r_{j,*}) \end{aligned}$$

#### Filter Functions

The task of a filter is to check whether a matrix or one of its elements fulfills certain conditions and to pass them on or sort them out accordingly. Thus a filter resembles an *if-then* statement. Describing this behavior by using mathematical functions requires the breakdown of the task and the establishment of some conventions. The defining part of each filter is its *condition*, which can be described in mathematical terms as function with the co-domain  $\{0, 1\}$ , representing the results false and true. A simple threshold condition, that is satisfied by all numbers smaller 10 could be defined as:

$$\begin{aligned} threshold : \mathbb{R} &\rightarrow \{0, 1\}, X \mapsto X^I \\ x^I &= \begin{cases} 1, & \text{if } x < 10 \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Adopting this notation for each condition would be pretty extensive, so we settle for a minimized version and only denote the condition leading to true resp 1 as the function name. Thus, notation of the preceding definition is reduced to  $\langle x < 10 \rangle$ .

With this convention, we have to look into the 'then' part of a filter. While elements that fulfill the provided condition are left

untouched, those who fail have to be sorted out or rather deleted. Actual deletion of elements or matrices cannot be modeled as mathematical function, therefore we need a workaround for this issue. Let us regard k-means as an example, where the minimal point-cluster distance is evaluated. Assume a row  $d_{2,*} = (d_{21}, d_{22}, d_{23}, d_{24})$ , from  $D$  whose components show the distances between point  $p_{2,ast}$  and the four centroids of  $R$ . The filter necessary for k-means requires to sort out all components that are not minimal. Without the capability for removal, it is necessary to define a neutral element to which all inputs that fail the condition are mapped. For our scenario, we state this neutral element as 0, which allows us to define the minimum filter as:

$$\begin{aligned} minFilter : \mathbb{M}^{1 \times k} &\rightarrow \mathbb{M}^{1 \times k} \\ (D, \langle x = min(D) \rangle) &\mapsto D^I \\ d_{ij}^I &= \langle x = min(D) \rangle (d_{ij}) \cdot d_{ij} \end{aligned}$$

Assuming  $d_{23}$  as minimum of  $d_{2,*}$ , the filtered row becomes  $d_{i,*}^I = (0, 0, d_{23}, 0)$ . Using this approach, the subsequent functions in a clustering algorithms have to be aware of 0 as neutral element. Our filter description is a composite of a variable condition and a fixed function that maps to the neutral element. In the context of clustering algorithms, filters cannot exist without conditions. However, conditions can exist by themselves and are necessary to describe branching and conditional execution of functions. Thus, standalone conditions are a way to realize control flow. The following pseudocode shows the notation for both cases:

---

$filter(M, \langle cond \rangle)$	$\triangleright$ input for filter
<b>if</b> $cond$ <b>then</b> function	$\triangleright$ standalone use

---

In both applications, the condition itself is denoted as boolean expression  $\langle cond \rangle$ , which is sufficient for its utilization as part of a filter. In standalone use a condition affects the control flow i.e. some actions are performed only if the condition is met. To illustrate this, we embed  $\langle cond \rangle$  in an if-then block, where the then part contains the action to be executed.

#### Association Function

By executing filters, a modified version of the input is created. For the selection phase, this is the modified distance matrix  $D^I$ , which is passed on to the association-function. The goal of this function is the transformation of distances into adjacencies i.e. the *point-distance-reference* triples that made it past the selection phase, must be converted into *point-reference* tuples. Basically  $D^I$  is converted into a binary matrix, where a value of 1 represents an existing adjacency. Due to the filtering, non-existent adjacencies have already been mapped to 0 which leaves the task of mapping every non-zero value to 1. Based on this, we define the binary association function *assoc* as

$$\begin{aligned} assoc : \mathbb{M}^{m \times n} &\rightarrow \mathbb{M}^{m \times n} \\ D^I &\mapsto A \\ a_{ij} &= sgn(d_{ij}^I) \end{aligned}$$

where  $sgn()$  is the sign function. This function is quite convenient as it keeps the neutral element 0 and maps all positive values to 1. Although  $sgn()$  can yield  $-1$  for negative inputs, this does not need to be considered in our setting as distances are always positive.



## 4.2 Control-Flow Structures

Until now we have not discussed one basic element which is crucial for almost every clustering algorithm but whose necessity is not evident. This additional basic element is the loop, which is a part of the control flow that we have not considered so far, with standalone conditions being the exception. The incorporation of loops in our scenario is tricky as they cannot be mathematically modeled, thus they are defined outside the mathematical domain. To describe clustering algorithms, we basically need two loop types: a *for-each* loop for element-wise traversal of datasets or clusterings and a *repeat-until* loop for conditioned iterations. These two loops are denoted with the following pseudocode:

---

```

for each element of M do
  (body)
end for  $\rightarrow M^I$ 

repeat with A
  (body)
until cond output  $\rightarrow B$ 

```

---

At the top, we find the block for the for-each loop, which is generally used to traverse datasets or clusterings by element. The opening statement of the loop specifies the traversed matrix  $M$  and the element/granularity of traversal: row, column or component. In our scenario, element-wise traversal is done by splitting up the source matrix into element-matrices—rows, columns or components at the beginning of the loop. After the split, the elements are processed individually according to the instructions of the *(body)*. As we want a single matrix as output again, the processed elements have to be re-assembled at the end of the loop e.g. row-matrices are appended. This re-assembly is implicitly assumed and not specifically noted in pseudocode. The loop output is denoted with the assignment after *end for*.

In addition to the described functionality, we use for-each loops for the actual removal of rows and columns from matrices. This is sometimes necessary when clustering algorithms delete references or clusters during optimization. With filters, we introduced mapping to the neutral element 0 as a means to tackle deletion. This works well and is also necessary for the clear definition of functions. However, the handling of whole rows and columns of zeros can become challenging e.g. it can lead to empty clusters in  $C$  or cause problems during the selection phase. Some of this issues could be tackled by introducing constraints to each function to ignore all-zero rows/columns. But this would be complex and not an overall solution. Our described for-each loop offers an elegant way to solve this problem. By inserting an appropriate condition before matrix reassembly at the end of the loop we prevent zero element-matrices from entering the output matrix. Since loops are outside the mathematic formalism anyway, adding row/column removal here provides us with a convenient tool without compromising the formal description of the remaining building blocks.

The *repeat-until* loop is used to represent conditioned loops. This kind of loops is normally used to control algorithm iterations, often during minimization/maximization of target functions like the sum of squared errors in k-means [10]. The stopping condition for the loop is always specified after the closing *until* statement. A repeat-until loop has one or more input matrices—denoted in the opening statement—which are continuously processed from iteration to iteration and an output matrix, obtained when the loop finishes. This output matrix can be either a processed version of the input or an assembly of element-matrices generated during the loop. The particular output type can be derived from the *(body)* of the loop.

## 4.3 Summary

In Section 2, we introduced the *core* of a clustering as a sequence of the phases *evaluation*, *selection* and *association* that acts as a general frame. Now that we finished the description of our building blocks and defined their syntax, we are able to concretize these phases and flesh out the mandatory algorithm *core*:

- **evaluation** - This phase requires at least 4 building blocks: one function playing the role of distance measure and three matrices acting as points, references and distances.
- **selection** - This phase consists of at least one filter or condition.
- **association** - For this phase 3 building blocks are mandatory: two matrices acting as adjacencies resp. clustering and the association function.

Beyond this essential structure, arbitrary clustering functionality can be added to each of these phases—including optimization—by utilizing the existing building blocks. We move on to the next section, where we demonstrate how clustering algorithms are transcribed using our proposed approach.

## 5. TRANSCRIPTION OF ALGORITHMS

This section demonstrates how clustering algorithms are transcribed using our approach, whereas we utilize two prominent clustering algorithms *k-means*[10] and DBSCAN[8]. While k-means is a representative of the clustering partitioning algorithm class, DBSCAN is from a completely different classed named density-based clustering.

### 5.1 k-means

Our representation of k-means is shown in Algorithm 1 and begins with the evaluation phase in which four building blocks take part. Three of these are actors: two matrices  $P$  and  $R$  that contain the points of the dataset and the  $k$  initial centroids as rows, as well as the distance matrix  $D$ . The fourth is the distance measure used to generate  $D$  from  $P$  and  $R$ . For k-means, this role is taken by the euclidean distance, denoted by the function  $L_2$  which we define for our matrix setting as:

$$L_2 : \mathbb{M}^{n \times f} \times \mathbb{M}^{k \times f} \rightarrow \mathbb{M}^{n \times k}$$

$$(P, R) \mapsto D \quad \text{with} \quad d_{ij} = \sqrt{\sum_{l=1}^f (p_{il} - r_{jl})^2}$$

where  $p_{i,*}$  and  $r_{j,*}$  are rows of their respective matrices. The resulting matrix  $D$  provides the input for the following selection phase, which starts with a for-each loop for row-wise traversal of  $D$  (6). Due to the evaluation phase, each row  $d_{i,*}$  contains all distances between point  $p_{i,*}$  and  $R$  that we need for the selection. The following filter function selects the minimum element  $d_{ij}$  from each row, which reflects the target function of k-means. At the end of the loop, the processed rows are assembled into the filtered matrix  $D^I$ , that is passed on to the association phase. There, our *assoc* function is deployed to generate the binary adjacency matrix  $A$ . Due to the character of k-means,  $A$  basically contains the final cluster assignments. As centroids resp. references represent the clusters,  $A$  is simply adopted as  $C$  (12).

With the core phases finished and a clustering result generated, k-means enters its optimization phase which updates the centroids (references) for the next iteration. Each centroid is recalculated

---

**Algorithm 1** k-means

---

```
1: repeat with  $R^x$ 
2:   phase EVALUATION
3:    $dist.L_2(P, R^x) \rightarrow D$ 
4:   end phase
5:   phase SELECTION
6:   for each  $d_{i,*}$  of  $D$  do
7:      $filter(d_{i,*}, \langle x = \min(d_{i,*}) \rangle)$ 
8:   end for  $\rightarrow D^I$ 
9:   end phase
10:  phase ASSOCIATION
11:   $assoc(D^I) \rightarrow A$ 
12:   $A \rightarrow C$ 
13:  end phase
14:  phase OPTIMIZATION
15:   $updt(C^T, P) \rightarrow R^{x+1}$ 
16:  end phase
17: until  $R^x = R^{x+1}$  output  $\rightarrow C$ 
```

---

as the arithmetic average of all points that were assigned to it in the current iteration. In our matrix-based setting, we realize this by using the matrix-multiplication as a template with  $C$  and  $P$  as input. The first input is matrix  $C$  that contains the point-cluster assignments and has the dimensions  $n \times k$  with  $k$  being the number of references/centroids. The second input  $P$  has the dimension  $n \times f$  with  $n$  being the number of points and  $f$  being the number of features of the dataset. By multiplying  $C$  with  $P$  we want to create an updated version of  $R$  having the dimension of  $k \times f$ . For this, the number of columns of  $C$  has to match the number of rows in  $P$ , which is not the case as  $k \neq p$ . Therefore, we transpose  $C$  to  $C^T$  which leads to the required column-row-match and results in a  $k \times f$  matrix  $R^{x+1}$  that contains the updated centroids for the next iteration. The function used for calculation of the update is defined as:

$$updt : \mathbb{M}^{k \times n} \times \mathbb{M}^{n \times f} \rightarrow \mathbb{M}^{k \times f} \quad (1)$$

$$(C^T, P) \mapsto R_{x+1} \quad (2)$$

$$r_{ij} = \frac{\sum_{l=1}^n c_{il} \cdot p_{lj}}{\sum_{l=1}^n c_{il}} \quad (3)$$

with  $c_{i,*}$  being a row of  $C^T$  and  $p_{*,j}$  being a column of  $P$ . Basically, this function uses each cluster represented by a binary row of  $C^T$  to select those values from the feature represented by  $p_{*,j}$  that belong to its members. This selection is summed up and normalized with the number of cluster members obtained by summing up all elements of binary  $c_{i,*}$ .

The whole algorithm is surrounded by a *repeat-until* loop that describes the iteration of k-means using the updated references/centroids of  $R_{x+1}$ . The stopping criterion, shown after *until* (17) is evaluated before a new iteration is started. For our depiction we choose  $R^x = R^{x+1}$  as stopping condition and quit the algorithm if the references/centroids no longer change, which indicates stabilized clusters. Of course other stopping conditions can be used e.g. reaching a fixed number of iterations.

## 5.2 DBSCAN

DBSCAN [8] is a density-based clustering algorithm that defines clusters as dense regions separated by regions of lower density. The algorithm uses two parameters  $\varepsilon$  and  $minPts$  to define a density threshold. With  $\varepsilon$  a neighborhood is defined around each point  $p$ . If this neighborhood contains at least  $minPts$  additional

points,  $p$  is considered as member of a dense area i.e. a cluster and is named *core-object*. Sets of core-objects with overlapping  $\varepsilon$ -neighborhoods are merged in order to create clusters. This is done recursively i.e. if  $p$  is a core-object each member of its  $\varepsilon$ -neighborhood is checked for the density condition.

---

**Algorithm 2** DBSCAN

---

```
1: phase EVALUATION
2:  $dist.L_2(P, R^x) \rightarrow D$ 
3: end phase
4: phase SELECTION
5:   for each  $d_{i,*}$  of  $D$  do
6:      $filter(d_{i,*}, \langle x < \varepsilon \rangle)$ 
7:      $sgn(d_{i,*}^I)$ 
8:      $filter(d_{i,*}^I, \langle \sum_{j=0}^n x \leq minPts \rangle)$ 
9:   end for  $\rightarrow D^I$ 
10: end phase
11: phase ASSOCIATION
12:  $assoc(D^I) \rightarrow A$ 
13:  $merge(A) \rightarrow C$ 
14:  $distinct(C) \rightarrow C_{distinct}$ 
15: end phase
```

---

The fully transcribed version of DBSCAN using our approach is shown in Algorithm 2. Although the evaluation phase may look the same as with the previously described algorithms, DBSCAN is different as it calculates the distances between all points, which means  $P$  and  $R$  are actually identical. The selection phase uses a for-each loop for row-wise traversal and contains three steps. First, a filter is employed to remove all distances that are bigger than the  $\varepsilon$ -neighborhood. Next  $sgn()$  is applied in preparation of the following filter, that tests if the neighborhood contains the necessary number of objects by checking the sum of components of the binary row-matrix. With the selection phase done, association starts with the known application of  $assoc()$  (12). After that, we face a challenge as  $assoc()$  effectively creates a cluster for each core-object and its  $\varepsilon$ -neighborhood.

Now, to determine the final clusters, overlapping  $\varepsilon$ -neighborhoods have to be merged. Utilizing recursion as proposed in [8] is not a valid approach in our matrix based setting. Therefore, we use a repeat-until loop to connect overlapping  $\varepsilon$ -neighborhoods as specified in Algorithm 3. For this, the adjacencies—labeled here as  $M^x$ —are multiplied with itself and the result is transformed into the binary  $M^{x+1}$ , which is the input for the next loop. With this, indirect/transitive cluster assignments are resolved. The loop ends if  $M^{x+1}$  does not change anymore, which means that all direct adjacencies have been found and the resulting clustering  $C$  is delivered.

---

**Algorithm 3** Transitive Merging Function:  $merge(M^x)$ 

---

```
1: repeat with  $M^x$ 
2:    $M^x \cdot M^x \rightarrow M^I$ 
3:    $sgn(M^I) \rightarrow M^{x+1}$ 
4: until  $M^x = M^{x+1}$  output  $\rightarrow M^{x+1}$ 
```

---

Example matrices for this association are shown in Figure 2, where the first three columns of  $M^x$  show the indirect cluster assignment of  $p_1, p_2$  and  $p_3$ . The matrix  $A = M^x$  is the result the selection phase and therefore, the input of the association function. Although the points  $p_1, p_2$  and  $p_3$  form a cluster, the adjacency of  $p_1$  and  $p_3$  is indirect via  $p_2$ . After multiplication, all adjacencies are explicit in  $M^{x+1}$ . While this solves the problem of merg-

$$\begin{array}{ccc}
\begin{array}{c} A = M^x \\ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \end{array} & \rightarrow & \begin{array}{c} M^I \\ 2 \ 2 \ 1 \ 0 \ 0 \ 0 \\ 2 \ 3 \ 2 \ 0 \ 0 \ 0 \\ 1 \ 2 \ 2 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 2 \ 0 \ 2 \\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\ 0 \ 0 \ 0 \ 2 \ 0 \ 2 \end{array} \\
& & \rightarrow \\
& & \begin{array}{c} M^{x+1} \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \end{array}
\end{array}$$

**Figure 2: DBSCAN association.**

ing overlapping  $\varepsilon$ -neighborhoods into clusters, it leads to duplicate columns/clusters. Complete explicit adjacencies mean that each member of a cluster is associated with each remaining member i.e. a cluster with 4 members manifests in 4 identical rows in  $C$ .

To get rid of the duplicates, we apply a duplicate elimination function *distinct* (14) on the output association matrix. The algorithm of this function is depicted in Algorithm 4. The function takes  $M^x$  as input and starts by aggregating it into a row matrix of cluster sizes  $H_{select}$ . Then, a maximum filter is deployed to select the biggest cluster and  $sgn(\cdot)$  is applied to make the resulting row-matrix  $H_{select}^I$  binary. This is only done to implement a processing sequence for the rows of  $M^x$ . Multiplication of  $H_{select}^{II}$  with  $M^x$  extracts a particular row matrix from  $M^x$ . This row matrix becomes the first row  $m_{i,*}$  of  $M_{distinct}$ . Example matrices for this are shown in Figure 3. After getting the first row, we have to get rid of all its duplicates in  $M^x$ . For this we apply  $sgn(\cdot)$  to  $m_{i,*}$  and create binary  $(m_{i,*})^T$ , whose transpose is multiplied with  $m_{i,*}$  to get a square filter matrix  $H_{filter}$ . By subtracting it from  $M^x$ , the processed row and its duplicates are set to zero, effectively removing them from further processing in the loop. The resulting  $M^{x+1}$  enters the next iteration, where another unique row is extracted. The loop ends when  $M^{x+1}$  becomes a zero matrix, which means all unique rows have been extracted. Examples of  $H_{filter}$ ,  $M^{x+1}$  and final  $M_{distinct}$  can be found in Figure 3, where the example output of the association phase in Figure 2 is continued.

---

**Algorithm 4** Distinct Function:  $distinct(M^x)$

---

```

1: repeat with  $M^x$ 
2:    $agg(M^x) \rightarrow H_{select}$ 
3:    $filter(H_{select}, \langle x = \max(H_{select}) \rangle) \rightarrow H_{select}^I$ 
4:    $sgn(H_{select}^I) \rightarrow H_{select}^{II}$ 
5:    $H_{select}^{II} \cdot M^x \rightarrow m_{i,*} \text{ of } M_{distinct}$ 
6:    $sgn(m_{i,*}) \rightarrow m_{i,*}^I$ 
7:    $(m_{i,*}^I)^T \cdot m_{i,*} \rightarrow H_{filter}$ 
8:    $M^x - H_{filter} \rightarrow M^{x+1}$ 
9: until  $\sum M^{x+1} = 0$  output  $\rightarrow M_{distinct}$ 
10: output  $M_{distinct}^T$ 

```

---

At the end of the *distinct* function, the result is transposed as we want clusters to be represented in columns. Afterwards, DBSCAN finishes as it has no optimization phase or global loop.

### 5.3 Summary

Due to the large number of approaches to clustering, we cannot transcribe each method or even a representative of each larger

$$\begin{array}{ccc}
\begin{array}{c} C = M^x \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \end{array} & & \begin{array}{c} H_{select} \\ 3 \ 3 \ 3 \ 2 \ 1 \ 2 \\ (b) \end{array} \\
& & \\
& & \begin{array}{c} H_{filter} \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ (d) \end{array} \\
& & \\
& & \begin{array}{c} H_{select}^{II} \\ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\ (c) \end{array} \\
& & \\
& & \begin{array}{c} M^{x+1} \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \\ (e) \end{array} \\
& & \begin{array}{c} M_{distinct} \\ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \\ (f) \end{array}
\end{array}$$

**Figure 3: Duplicate Elimination.**

family of algorithms in this paper. However, we are sure that our approach is general enough to cope with this variety. Often it seems that a particular algorithm cannot be transcribed at first, but after re-considering a way can be found to adapt to our setting. Sometimes, little modification are enough, but in other cases the problem must be rethought thoroughly to find an equivalent building block representation. Examples include but are not limited to:

- Graph-clustering, where graphs must be transformed into a matrix for adaptation.
- Evolutionary approaches e.g. artificial immune systems [18], model centroids or proto-clusters as a population of cells that is influenced by a fitness function. This can be modeled in our approach by modifying references/clusters between iterations. Creation, deletion, re-calculation as well as splitting and merging, can be reproduced with modified matrix multiplications, filters and custom mathematical functions.
- Spectral clustering approaches, work with the eigenvalues of a similarity matrix. Although it seems that this resembles our distances  $D$ , this is not the case. As partitioning is based on the correspondence between eigenvalues, this measure must be considered during evaluation. In this case the initial similarity matrix must be seen as an additional input.

This should only point out the versatility of our approach. We are pretty sure that there are still a lot of further clustering approaches, that we did not consider in detail. But we are also sure that we could describe them with our approach after giving them some thought. To summarize our approach offers the following advantages:

1. Easy adaptation and specification of clustering algorithms in an abstract and implementation-independent way. In general, our approach is designed for the data scientist.
2. The comparison of algorithms and the easy identification of common functionalities.

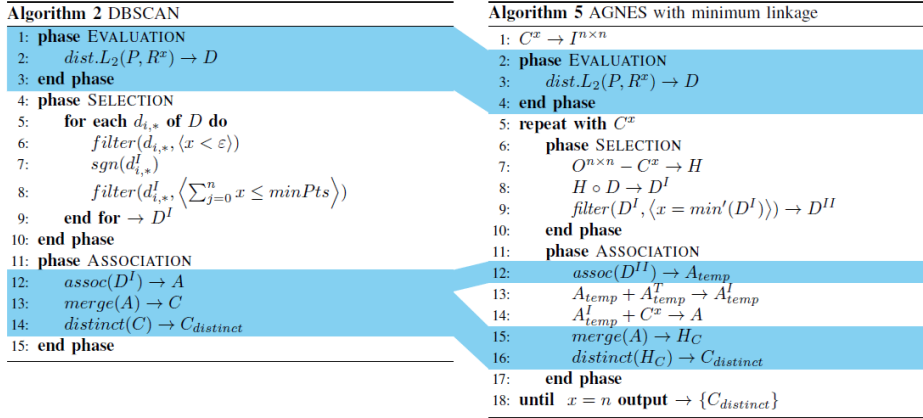


Figure 4: Similarities of DBSCAN and AGNES.

## 6. CONTRIBUTIONS

Besides providing a way to specify clustering algorithms, our building blocks approach offers several novel application possibilities. Due to the use of consistent components, we can evaluate the similarity between algorithms to a certain degree. During the transcription of our example algorithms, we uncovered the existence of several description blocks that are used by multiple algorithms e.g. removal of duplicates and resolution of transitivity. Fig. 4 shows a comparison of the already described DBSCAN and AGNES[12, 13] as an example from a further hierarchical clustering algorithm class, where identical building blocks are highlighted in blue. It is easy to see, that both methods are very similar. Evaluation phases are identical, while the association phases show only minor differences in two lines that are necessary because each  $C$  generated by AGNES is also considered a hierarchy level. The main differences between both methods are located in the selection phase and during optimization. While this allows the easy identification of an algorithms characteristic parts, it can also be used to classify algorithms. Families of algorithms that share certain traits could be identified on the basis of commonalities. In order too find common patterns, frequent itemset mining could be applied to a repository of algorithms. For this, each building block is considered as an item and each phase resp. algorithm as transaction.

Each of our phases and building blocks encapsulates a defined functionality. Furthermore, we observed that certain blocks can be used in different algorithms. Therefore, we do not limit our concept to the description/translation of existing methods but also use it for the creation of new ones. The modular character of our approach enables algorithm creation in a novel and easy way. Basically, there are three levels of modularity that can be used to build new algorithms and are depicted in Fig. 5.

**1st Level: Phase-Swap** Phases realize the basic tasks necessary for clustering in an algorithm-specific way. Although phases are implemented individually, they share a defined interface. This means, evaluation always produces a distance matrix  $D$ , selection always uses  $D$  and creates  $D^I$ , and association always creates  $C$  from  $D^I$ . With this, phases become interchangeable and form the largest modules of our approach. The optimization phase is more individual, but can still be swapped if only the mandatory elements  $P, R, D, D', A, C$  are accessed. All this allows users to easily create new algorithms by recombining phases of available stock algorithms.

**2nd Level: Custom Phase** This level works on the finer granularity of building blocks and enables intermediate users to create new evaluation, selection, association, and optimization phases from scratch. For this, existing blocks from a repository are combined and fitted into our introduced phase-templates. Examples for such blocks are  $updt()$  and  $distinct()$  from our example algorithms section. Newly created phases can be added to the existing repository and thus provide new options for level 1.

**3rd Level: Custom Block** On this level, experienced user can freely define new building blocks e.g a new distance function or a scenario specific variant of  $assoc()$ . In addition, block sequences or subroutines that occur very often, can be integrated as higher-order building blocks to ease description. Like before, new building blocks are added to a repository, where they are available for other users. The creation of new building blocks also makes the previous levels more versatile.

The first and second level implement creation exclusively by combination of modules/blocks from a repository. This makes it possible to realize this task with interactive interfaces instead of IDE's normally used for software development. In Fig. 6, we depict a prototypical interface for modular algorithm design that shows k-means by using our building blocks as basis for the visual elements. Each phase is marked by a different color: blue for evaluation, green for selection and so on. This prototype was designed for smart devices, and allows users to swap phases by swiping and switch building blocks by touching. If one of these actions is performed, the system provides a list of alternatives, available in the repository, from which the user can choose.

## 7. FUTURE WORK

Looking back at our motivation, we argued that the ever increasing diversity of clustering algorithms is necessary to cope with the individual characteristics of various data sets. Because, this diversity complicates the application of clustering in the context of Big Data, a building block approach would be desirable and was proposed in this paper. Aside from supporting the specification, our approach facilitates the adaptation of core clustering principles for specific applications.

Generally, our modular approach consists of a few functions like  $assoc$ ,  $merge$ ,  $updt$  and  $filter$  over a single data structure of type  $matrix$ . Our next research step focuses on a general mapping of these functions to a MapReduce infrastructure. In this step, the

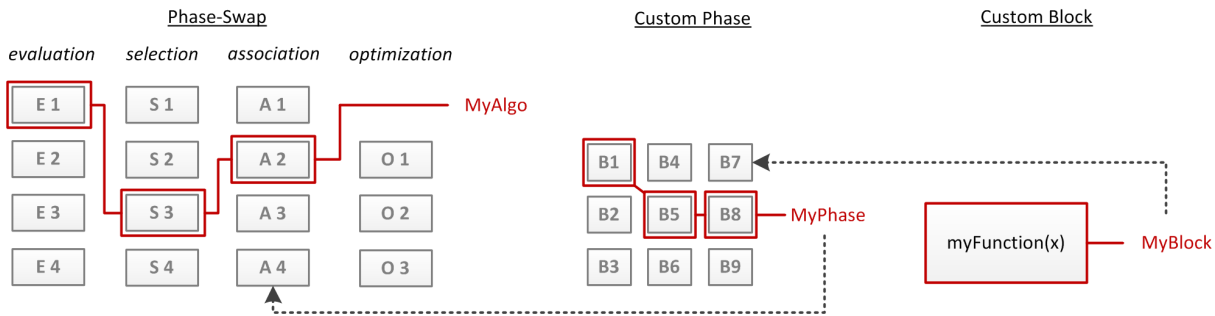


Figure 5: Levels of modular algorithm design.

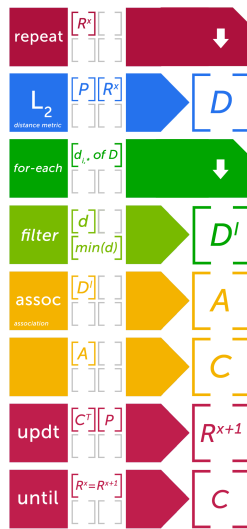


Figure 6: Prototype UI for modular algorithm design.

following challenges arise:

- As described in Section 4, the *repeat-until* loops are integral part of our building blocks. Therefore, we require a MapReduce infrastructure with support for iterative computations as proposed in [4].
- The most challenging issue is the mapping of our matrices to  $(key, value)$  pairs in MapReduce. This mapping has to be done, so that an efficient and scalable partitioning is possible in order to parallelize necessary function evaluations. In [14], we investigated this aspect in MapReduce and presented several approaches, whereas there exists no best fitting approach for all cases. The best-fitting approach depends on several properties like matrix size or number of available nodes.
- Aside from mapping of our matrix construct, we also have to specify physical operators for our limited set of logical functions.

As a result of this work, we are able to transform any arbitrary clustering algorithm specified in our building blocks into an efficient and scalable execution form. Furthermore, we can optimize

the transformation by the utilization of different mapping strategies of matrices to  $(key, value)$  pairs as well as selection of the best-fitting physical operator for a logical building block function. For this optimization, we have to define various optimization strategies depending e.g., on matrix sizes or sequence of logical functions. Furthermore, our iterations can be integrated in the optimization, since our loops iterate either row- or column-wise over the matrix. In this way, we establish a similar approach as conducted in database systems for over 30 years with SQL as logical interface and for each logical operator different physical operators exist. The transformation between the logical and physical layer is done using an optimizer component, which is responsible for efficient transformation using roles and a cost-model. In our next research step, we want to establish such an approach for data clustering algorithms in highly scalable parallel data processing platforms.

A second major approach to execution is direct integration into the database. Already, approaches like SciDB [5] consider matrices as first-order citizens in database management systems. This way of integration is especially compelling as it allows a tight coupling of data management and data analysis in the scope of a single platform. Besides the deployment to different software systems, also hardware specifics can be considered. By developing platform-specific compilers for e.g. NUMA or GPU-centric systems, users could create optimized versions of their algorithm libraries in an on-demand fashion. In this case, a lot of related research is and has been done in different domains, that can be used in our future work. For example, efficient large matrix computation has a long tradition as area of research in high performance computing [11, 15]. Furthermore, graphic cards and CUDA are strongly geared to matrix processing [1, 9, 17] and seem to be an ideal target architecture our approach. In this domain, dense [1, 19] as well as sparse [3] matrix operations are well-investigated.

## 8. RELATED WORK

On the one hand, our work is motivated by SciDB [5], which only considers the storage and the processing of a natural nested multi-dimensional array data model. On the other hand, our approach originates from the ongoing *key-value* hype of MapReduce [7], because from our point of view the *key-value* model is not appropriate for the data clustering domain. As shown in [20, 2], MapReduce and an enhanced paradigm based on a *key-value* data model can be used for the *k-means* clustering algorithm. However, the implementation of essential clustering functionality is complicated by the restricted data model. Several approaches are proposed to map the necessary matrix data to a *key-value* model. One possibility is to encrypt row and column information in the key forming a super-key. Nevertheless, a pure matrix model as proposed in our approach is more direct and eases the specification of clustering algorithms.

In a derived implementation out of our modular algorithm specification, we are able to use the *key-value* data model for scalable execution.

In the context of highly scalable parallel data processing platforms, the Mahout<sup>3</sup> project has to be considered, which directly implements various machine-learning algorithms in Hadoop. The implementations currently neither exploit high-level data processing languages built on top of Hadoop nor do they make use of any statistical software. With more and more analysis methods are added, leveraging existing functionality adds to the stability and simplicity of the implementation. Instead of implementing and optimizing each single analysis method separately, our approach introduces a novel abstraction layer on top to specify methods in an implementation-independent way using building blocks. The building blocks have to be mapped to the execution unit, e.g. Hadoop once and each algorithm can directly benefit. The mapping of our building blocks is our next step as described in the previous section.

## 9. SUMMARY AND CONCLUSION

In this paper, we proposed our modular building blocks approach as a unified construction kit for clustering algorithms. We decomposed clustering methods and derived the core of every algorithm in the form of the three phases: evaluation, selection and association. In addition with the optional optimization phase, this provides a general frame for algorithm description. To fill this frame, we identified the basic elements of each phase and transformed them into a set of building blocks. In our data model, all necessary objects for clustering i.e. points, references, distances, adjacencies and clustering are formally represented as matrices. Matrices are the exclusive and universally valid way for data modeling in our approach and naturally match the concept of clustering. Based on this, all necessary operations like distance measurement, filtering and association are modeled as mathematical functions on matrices. To complete our set of building blocks, we introduced conditions and loops to represent the control-flow of a clustering algorithm.

All this was put to use during the transcription of k-means and DBSCAN which are well-known members of the two major classes of clustering algorithms. Our transcription proved that different methods can be easily represent with our unified description. Furthermore, our approach allows the comparison of algorithms and the easy identification of common functionalities. Besides this benefits for understanding, adaptation and construction of clustering algorithms, our descriptions can be used as a starting point for platform-specific implementation. From our point of view, this offers considerable potential for the efficient execution of any clustering algorithm.

## 10. REFERENCES

- [1] S. Barrachina, M. Castillo, F. D. Igual, R. Mayo, E. S. Quintana-Ortí, and G. Quintana-Ortí. Exploiting the capabilities of modern gpus for dense matrix computations. *Concurrency and Computation: Practice and Experience*, 21(18):2457–2477, 2009.
- [2] D. Battré, S. Ewen, F. Hueske, O. Kao, V. Markl, and D. Warneke. Nephelē/pacts: a programming model and execution framework for web-scale analytical processing. In *SoCC*, pages 119–130, 2010.
- [3] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, November 14-20, 2009, Portland, Oregon, USA, 2009*.
- [4] Y. Bu, B. Howe, M. Balazinska, and M. D. Ernst. Halooop: Efficient iterative data processing on large clusters. *PVLDB*, 3(1):285–296, 2010.
- [5] P. Cudré-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla, D. J. DeWitt, B. Heath, D. Maier, S. Madden, J. M. Patel, M. Stonebraker, and S. B. Zdonik. A demonstration of scidb: A science-oriented dbms. *PVLDB*, 2(2):1534–1537, 2009.
- [6] S. Das, Y. Sismanis, K. S. Beyer, R. Gemulla, P. J. Haas, and J. McPherson. Ricardo: integrating r and hadoop. In *SIGMOD Conference*, pages 987–998, 2010.
- [7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
- [8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231, 1996.
- [9] K. Fatahalian, J. Sugerman, and P. Hanrahan. Understanding the efficiency of gpu algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware 2004, Grenoble, France, August 29-30, 2004*, pages 133–137, 2004.
- [10] E. W. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, 21, 1965.
- [11] K. Goto and R. A. v. d. Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3):12:1–12:25, May 2008.
- [12] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.
- [13] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.
- [14] T. Kiefer, P. B. Volk, and W. Lehner. Pairwise element computation with mapreduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10*, pages 826–833, 2010.
- [15] M. Krishnan and J. Nieplocha. Memory efficient parallel matrix multiplication operation for irregular problems. In *Conf. Computing Frontiers*, pages 229–240, 2006.
- [16] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon. Parallel data processing with mapreduce: a survey. *SIGMOD Record*, 40(4):11–20, 2011.
- [17] S. Ohshima, K. Kise, T. Katagiri, and T. Yuba. Parallel processing of matrix multiplication in a cpu and gpu heterogeneous environment. In *High Performance Computing for Computational Science*, pages 305–318, 2006.
- [18] J. Timmis, M. Neal, and J. Hunt. An artificial immune system for data analysis. *Biosystems*, 55(1&A3):143 – 150, 2000.
- [19] V. Volkov and J. W. Demmel. Benchmarking gpus to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing, SC '08*, pages 31:1–31:11, 2008.
- [20] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. In *Proceedings of the 1st International Conference on Cloud Computing*, pages 674–679, 2009.

<sup>3</sup><http://mahout.apache.org>

# Bidirectional Transformations (BX)

Soichiro Hidaka (National Institute of Informatics, Japan)  
James Terwilliger (Microsoft, USA)

# Preface to the Third International Workshop on Bidirectional Transformations

Soichiro Hidaka  
National Institute of Informatics, Japan  
hidaka@nii.ac.jp

James F. Terwilliger  
Microsoft Research  
james.terwilliger@microsoft.com

## ABSTRACT

This workshop is the third in a series promoting cross-collaboration between computer science disciplines on the topic of bidirectional transformations. The workshop had a 53% acceptance rate from submissions from four different disciplines. In this brief preface, we outline a definition for what makes a bidirectional transformation, and describe a history of the workshop and its associated meeting series.

## Keywords

Bidirectional Transformations, Cross-Disciplinary

## 1. INTRODUCTION

The workshop series on Bidirectional Transformations (BX) is dedicated to bringing researchers together from different disciplines of computer science working on BX-related projects. There are at least four disciplines actively or historically working on such projects:

1. *Databases*, whose history of work on updateable views [2, 4] often serves as the semantic backbone for other work both within and without the field, and whose work on data exchange represents an opportunity to apply such theory to new applications [1].
2. *Programming languages*, whose work on lenses [5] often form the formal basis for new work.
3. *Graph transformations*, whose work on triple graph grammars [8] serves as a way to grow transformations from simple rules.
4. *Software engineering*, whose work stems from a need to manage significant numbers of software artifacts in practical settings.

We accepted 9 out of 17 submissions. Each of the four sub-disciplines was represented by at least one submitted paper.

Previous workshops were held with the European Joint Conferences on Theory and Practice of Software (ETAPS),

a federation of smaller conferences tailored to several disciplines of computer science but not a database audience. As the intent of the workshop series is to rotate through venues to promote communication across disciplines, the steering committee made the decision to hold this year's instance in association with a database conference for the first time.

The next BX workshop will likely be held with the Software Technologies: Applications and Foundations (STAF) federation of conferences. Among the many conferences associated with STAF is the International Conference on Model Transformations (ICMT). ICMT had often been a frequent venue for papers on BX, especially prior to the start of the workshop series, so associating with STAF next year will be a good fit for the workshop and a way for the BX workshop to return home for a year, after a fashion.

## 2. WHAT IS BX?

Bidirectional transformations are a mechanism to maintain consistency between two (or more) related sources of information [3]. One can think of a bidirectional transformation as a pair of transformations in opposite directions. Suppose one is given two sets  $S$  and  $T$  of artifacts such as strings, trees, or tables. For a given source artifact  $s \in S$ , the *forward transformation*  $f : S \rightarrow T$  produces a target artifact  $t \in T$ , while a *backward transformation*  $g : T \rightarrow S$  produces a source from a target. To achieve consistency using the bidirectional transformations,  $f$  and  $g$  are required to satisfy certain round-tripping properties.

The obvious case is when  $g = f^{-1}$ . However, this case is often considered to be too restrictive. In particular,  $f$  is not allowed to lose information. To overcome this restriction,  $g$  is allowed to access the original source artifact, making the function binary, i.e.,  $g : S \times T \rightarrow S$ . This round-tripping property, or *well-behavedness*, can be characterized as follows, giving names *get* and *put* to the forward and backward transformations [5]:

$$\begin{aligned} \forall s \in S. \text{put}(get(s), s) &= s && \text{(GetPut)} \\ \forall s \in S, \forall t \in T. get(\text{put}(t, s)) &= t && \text{(PutGet)} \end{aligned}$$

GetPut says that you can come back to the original source if there is no change on the target, while PutGet says that all information in the target are propagated to the source (therefore the *get* can always recover the updated target).

It is worth noting that the above formulation corresponds to the properties for the view update problem [2]. The view update problem is, given a database state  $s \in S$  and a query  $f : S \rightarrow T$  and an update  $u : T \rightarrow T$  on the view  $f(s)$ , translating  $u$  to the update  $T_u : S \rightarrow S$  on the database.



Then one can formulate the desirable properties as:

$$\forall s \in S. uf(s) = f(s) \Rightarrow T_u(s) = s \quad (\text{Acceptability})$$

$$uf = fT_u. \quad (\text{Consistency})$$

Acceptability says if there is no update on the view, then there will be no update on the database, corresponding to GetPut. On the other hand, Consistency says the updated view  $uf(s)$  can be reconstructed by regenerating a view from the database on which the translated update  $T_u$  has been applied. Therefore it corresponds to PutGet.

The scheme above is asymmetric in that *put* was binary while *get* remained unary. In graph transformation and its application to software engineering, we often see a symmetric scheme where the old target is used in the forward transformation. Consistency is no longer represented by a function, but by a relation  $R \subset S \times T$ .  $(s, t) \in R$  if and only if  $s \in S$  and  $t \in T$  are consistent. If the forward and backward transformation are denoted by  $\vec{T}$  and  $\overleftarrow{T}$  respectively, then well-behavedness is expressed by:

$$(s, t) \in R \Rightarrow \vec{T}(s, t) = t \wedge \overleftarrow{T}(s, t) = s \quad (\text{Hippocraticness})$$

$$(s, \vec{T}(s, t)) \in R \wedge (\overleftarrow{T}(s, t), t) \in R. \quad (\text{Correctness})$$

Hippocraticness says that if the forward transformation is applied to an already consistent pair of source and target, the same artifact (original target for  $\vec{T}$ , or original source for  $\overleftarrow{T}$ ) is obtained, so it corresponds to GetPut in the asymmetric setting. On the other hand, Correctness says that the result of the transformations are always consistent. It corresponds to PutGet in the sense that for updated target  $t$ , source  $s$  that satisfies  $get\ s = t$  is always obtained.

Different approaches and implementations often refer to different notions of correctness properties, and community-wide efforts have been made to share the notions with examples, some of which also appear in this volume.

### 3. HISTORY AND CONTEXT

Since 2008, researchers from the four disciplines referenced in Section 1 have been meeting periodically with the intention of opening up communication between those fields and potentially establishing a common research agenda.

#### *Shonan: December, 2008*

The first meeting was in Shonan near Tokyo in 2008 [3] and served primarily as an introduction. At that time, most of the participants had little-to-no exposure to the research going on in other disciplines. Most of the meeting was spent with participants introducing their own work, or relevant research with which they are familiar. By the end of the week, the participants collectively decided that there was significant overlap between that work, enough to merit further discussion. They decided that such work should be citing each other more, and there could be some interesting collaboration and unification to be done. Work began to arrange another meeting to follow up on possible collaborations.

#### *Dagstuhl: January, 2011*

A second meeting occurred in early 2011 at Dagstuhl [6]. The meeting began with representatives from each discipline giving short-form tutorials of 2–3 hours on the tools from that discipline to bring people up to speed quickly. The tutorials presented not only the bidirectional problems

intrinsic to that discipline and the primary tools in its solution space, but also that discipline’s requirements for formal properties of BX. There was ample time for new participants to present their own work. Finally, there was space in the schedule left open for group work and discussion.

The primary work product of the Dagstuhl meeting was the establishment of the BX workshop series itself, whose first instance was the following year in Tallinn, Estonia.

#### *Banff: December, 2013*

Most recently, another meeting was held in late 2013 in Banff, Canada (summary publication forthcoming). The meeting contained some short discipline-specific tutorials again, but most of the time was dedicated to working group and breakout sessions. These sessions covered a number of topics, but two of the most populated and productive focused on how to benchmark BX tools and how to put together a repository of examples. Both of those discussions yielded work that is published in this workshop proceedings.

What has become clear over the course of the meetings is that it is difficult in any collaboration to get past the point where people primarily talk about their own work. Defining a metric of success for this sequence of meetings — and this workshop series as well, for that matter — is difficult, but almost certainly must include an increase in cross-disciplinary citation rates. It is not necessarily the case that we will eventually arrive at a central, unified research agenda, but at the very least we hope to see far more opportunities for collaborative research and publications (e.g., [7, 9]).

Another meeting is being planned at the time of this publication. It will likely happen sometime in 2015 or 2016. Most of the details are still in development.

### 4. REFERENCES

- [1] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. Relational and XML Data Exchange. Morgan and Claypool Publishers, 2010.
- [2] Bancilhon, F. and Spyrtos, N.: Update Semantics of Relational Views, *ACM Transactions on Database Systems*, December 1981, 6(4).
- [3] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. *ICMT 2009*.
- [4] U. Dayal and P. Bernstein. On the Correct Translation of Update Operations on Relational Views. *ACM Transactions on Database Systems*, September 1982, 8(3).
- [5] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. *POPL 2005*, 233–246.
- [6] Z. Hu, A. Schürr, P. Stevens, and J. F. Terwilliger. Dagstuhl seminar on bidirectional transformations (BX). *SIGMOD Record* 40(1), (2011).
- [7] M. Johnson, J. Pérez, and J. F. Terwilliger. What Can Programming Languages Say About Data Exchange? *EDBT 2014*.
- [8] A. Schürr. Specification of graph translators with triple graph grammars. *WG '94*, pages 151–163. June 1995.
- [9] J. F. Terwilliger, A. Cleve, and C. Curino. How Clean Is Your Sandbox? *ICMT 2012*.

# Implementing a Bidirectional Model Transformation Language as an Internal DSL in Scala

Arif Wider

Humboldt-Universität zu Berlin  
Unter den Linden 6  
Berlin, Germany  
wider@informatik.hu-berlin.de

Beuth Hochschule für Technik Berlin  
Luxemburger Strasse 10  
Berlin, Germany  
awider@beuth-hochschule.de

## ABSTRACT

Despite advantages in terms of comprehensibility, verification, and maintainability, bidirectional transformation (bx) languages lack wide-spread adoption. Possible reasons are that tool support for bx languages is sometimes weak or outdated, that many bx languages are hard to integrate with existing software technologies, or that bx languages often cannot be mixed with unidirectional transformation languages and general-purpose programming languages.

We present an approach to implement existing bx languages as *internal domain-specific languages* (iDSLs) in the Scala programming language and demonstrate the approach by implementing *state-based tree lenses* as a statically typed iDSL in Scala. We show that this approach allows for rich tool-support based on static analysis and for achieving technological integration with the Java platform in general and with the Eclipse Modeling Framework (EMF) in particular. At the same time, the iDSL is independent from DSL-specific tool-support and can be mixed with Scala, Java, or unidirectional transformation languages.

## 1. INTRODUCTION

Most bidirectional transformation (bx) languages are *external* domain-specific languages (DSLs), i.e., they come with their own tools – e.g., for parsing, editing, or verification – which were specially developed for them. Creating good tooling for a bx language takes a lot of effort. Furthermore, the tooling has to be maintained, e.g., in order to stay compatible with other software development tools. For bx languages, this is especially a problem, because they have (so far) only a limited user base, and thus, it is hard to justify (or finance) putting lots of efforts into their tooling. However, if the tooling is not good or kept up-to-date, bx users will fall back to use better supported unidirectional transformation languages or even general-purpose languages (GPLs) like Java for implementing their synchronizations (despite disadvantages concerning verification, comprehensibility, or maintenance).

An alternative approach are *internal* DSLs (iDSLs): An iDSL is basically a library, written in a *host language* which is usually a GPL. However, in host languages that provide powerful abstraction concepts and/or a flexible syntax, one can create libraries whose look and feel get close to that of an external DSL. The main advantage of an iDSL is that, naturally, the tooling of the host language can be reused without modification. Of course, the tool support of an external DSL is potentially more powerful, as it can be tailored to the DSL. E.g., error messages of iDSLs are often hard to understand. However, good generic tool support especially of statically-typed GPLs (e.g., debugging, static analysis, etc.) can go a long way for using an iDSL comfortably.

In this paper, we show how an existing bx language – state-based tree-lenses as presented by Foster, Pierce, et al. in [3] – can be implemented as an iDSL in Scala. We show how this way, tree lenses (a combinator-based, asymmetric bx language) can be adapted to work in an object-oriented context and how they can be integrated with existing Java-based technologies like EMF. Our approach mainly relies on the pre-assumption that models are graphs which always have a spanning containment tree; an assumption that is true for many modeling technologies in general, and for EMF in particular. The technological integration mainly relies on Scala being a JVM-language that supports both object-oriented and functional programming. Thus, Scala is well-suited for integrating functional programming techniques with object-oriented concepts and with Java-based technologies. Lenses defined with our iDSL can directly process the Java-instances that represent an EMF model at runtime. Furthermore, we use the Scala compiler to perform extensive static type-analysis using the type information provided by the Java classes which are generated from an EMF metamodel. This way, the corresponding error highlighting, syntax checks, and code completion features can be provided by any Scala IDE plug-in and no further tooling is needed. The paper is structured as follows: The next section introduces Scala concepts which are important for our iDSL. Sec. 3 presents the data model that our iDSL uses and Sec. 4 explains how we convert models accordingly. Sec. 5 shows how we achieve type-safety and Sec. 6 demonstrates the iDSL with an example. Related work concludes the paper.

## 2. IMPORTANT SCALA CONCEPTS

Scala programs are compiled to regular JVM bytecode. Therefore, one can access Java code from a Scala program and vice versa. Also, Scala’s syntax is intentionally close to that of Java. Notable exceptions are that (1) type anno-

tations follow identifiers, separated by a colon, (2) type parameters are enclosed in square brackets, and (3) line-ending semicolons as well as dots and parentheses for method invocation are often optional. Furthermore, because Scala supports type-inference, type annotations can often be omitted as shown in the following listing:

```
val x:Int = "1234".length(); is similar to val x = "1234" length
```

An important concept that we make extensive use of in our iDSL are *implicit conversions*: When marking a function as implicit, the Scala compiler will automatically insert calls to that function if this can solve a compile-error:

```
class RichString(str: String){ // a class that wraps a string
  def mylength = str.length //...and provides additional methods
}
implicit def string2richString(str:String) = new RichString(str)
//because of the above implicit conversion this code compiles:
val x = "1234".mylength // ...as it is implicitly augmented to:
val x = string2richString("1234").mylength
```

The other Scala concept that we make extensive use of, are *type members*: In Scala, a class can have types as members, too. The following listing shows a class with a (1) type parameter `T` – which must be a subtype (denoted by `<:`) of type `AnyVal`, i.e., a primitive or value type like `Boolean`, `Int`, etc. – and (2) a type member `ElementType`.

```
class ValueList[T <: AnyVal](lst: List[T]) {
  type ElementType = T
}
```

Here, type member `ElementType` holds the type with which the class is parameterized. It can either be accessed via an instance, e.g., `myvalueList.ElementType`, or via a parameterized type, e.g., `ValueList[Int]#ElementType`. Now, because type members can have type parameters themselves, one can define type functions which are evaluated at compile-time. As type members can also be abstract, they can be declared in supertypes and implemented in subtypes. The declaration of an abstract type function equivalent to `def f(x: Dom) : Cod` would be `type F[X <: Dom] <: Cod`.

### 3. A DATA MODEL FOR LENSES IN SCALA

For implementing tree lenses in Scala, we need to adapt the data model of the original state-based tree lenses – edge-labeled trees – in order to successfully apply the lens combinator concept to an object-oriented, JVM-based setting: An object is a triple of a unique identity by which it can be referenced, a state, and a class that defines valid operations on that object. The state of an object consists of the values of a fixed number of fields. In a Java-based context, fields have a unique name and a static type. Fields containing multiple values can be expressed as a homogeneously typed collection, e.g., an indexed list or a key-value map. In contrast, figure 1 shows how data is represented in the edge-labeled tree data model of the original tree lenses.

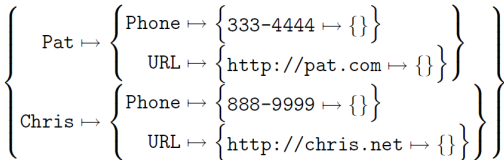


Figure 1: An address book as an edge-labeled tree

In the edge-labeled tree, labels are used to access the children of a tree node. The counterparts in objects are either field names or the index (or key) by which one can access a

specific element in a collection. Now, whereas in the edge-labeled tree data is stored as labels – e.g., the phone number in the example – we cannot save data as a field name in Java or Scala. This reveals one of the main differences between the original data model and the needed one: With the edge-labeled tree, we have no meta-layer but only an instance layer, i.e., both meta-information (e.g., description of the contents of a field) and value information is mixed (both "phone" and "3334444" are labels). So, what is always a label in the edge-labeled tree, is in object-oriented terms sometimes meta-information and sometimes value-information. This means, we need both lenses that work on the meta-level and lenses that work on the instance/value level.

Another difference is that objects can reference other objects which are not considered their children, i.e., they can have non-containment references, and thus, object structures (and models) are graphs. However, if we look at Java-based application frameworks like EMF, it is characteristic that a spanning containment tree is enforced, i.e., object structures must have an explicitly marked root-object and objects can have at most one container. This constraint has been shown to be very useful, e.g., for fast graph traversal and persistency management. Thus, in practice, many object structures are graphs with an underlying spanning containment tree. We rely on this constraint to pragmatically apply tree lenses to an object-oriented context.

Finally, the children of a tree node in an edge-labeled tree are *unordered* and can be *non-unique*. Now, as we defined field names or collection indices/keys, respectively, as the counterparts to labels for accessing child elements, children are unique: indices or keys are unique by definition and field names in Java/Scala are also required to be unique in one class. Concerning order the situation is more diverse: indices are obviously ordered but class fields and dictionary keys are generally considered unordered. However, EMF for instance, represents children of a tree node as an ordered list for XML persistence reasons. Furthermore, the fields of Scala case classes coincide with the parameter list of the class' constructor which is ordered. Thus, for uniformity, we define the 'labels' to access the children of a tree node as an ordered list without duplicates. Note that the uniqueness constraint only applies to the labels to access the children, thus, there can be duplicate elements in a list as the indices are unique. Furthermore, we do not represent tree leafs using empty child lists, but by special value tree node types. We call this data model, which we designed as a pragmatic adaptation of an edge-labeled tree for an object-oriented context, an *object tree*:

*Definition 1.* An *object tree*  $\mathcal{T} = \langle t, id, [v|l] \rangle$  is a triple of a type-annotation  $t$ , a unique identity  $id$ , and either a single value  $v$  or an ordered list  $l$  referring to either a fixed number of subtrees (the fields) or an arbitrary number of subtrees of the same type (the elements of a collection). Single value tree nodes can represent a non-containment reference by holding the id of another tree node.

We implemented this data model as a Scala class type hierarchy with an abstract root type `Term` (any tree node) and several subtypes, e.g., for list terms, tuple terms, constructor terms, value terms, and reference terms. Together with type annotations, this allows us to express type constraints on the data that a lens can handle. Comparing this data model with the one of tree lenses, type-annotations and object-ids were added, and order of subterms now matters.

Edge-labels are replaced by indices which – in the case of a constructor term – can be mapped to field names (using the type annotation). This data model allows us to implement most of the original tree lenses with similar semantics for model transformations but also allows for defining special lenses for an object-oriented setting.

## 4. IMPLICIT CONVERSION BETWEEN MODELS AND TYPED TERM TREES

In the following three subsections, we show (1) how to convert a domain object (i.e., a model element) to a typed term, (2) how to convert a model to a tree with cross-references, and (3) how to ensure referential integrity in this conversion.

### 4.1 Converting Domain Objects to Typed Terms

In order to be able to implement a set of pre-defined lenses (i.e., a lens library / lens language) independently from specific domain classes, lenses need to be defined against general term types. However, to apply these lenses directly on domain objects, domain objects have to be converted to terms. We use Scala’s *implicit conversions* for transparently converting domain objects to terms and vice versa. We want to preserve static type-safety throughout the whole transformation process. Therefore, we have to keep track of the types of all of a term’s subterms. This cannot be achieved by annotating terms with a corresponding class type, because in the transformation process *intermediate structures* can emerge that do not correspond to any source or target domain class (e.g., when splitting up a source domain object, the results of this splitting need to get a type, before putting them together to a target domain object).

Because Scala’s type system – and other common type systems – only provide either a heterogeneously typed tuple construct with a fixed arity (e.g., `Tuple3[A,B,C]`) or a homogeneously typed collection (e.g., `List[A]`), we use *heterogeneously typed lists* (HLists), as introduced for Haskell by Kiselyov et al. [6], as the underlying data structure. HLists are based on type-parameterized, nested *Cons-cells*. This way, heterogeneously typed list instances can be defined with static type-safety. However, in code both nested type annotations and nested list instantiations are verbose and error-prone. Therefore, some Scala implementations of HList<sup>1</sup> define a *typelist* type (TList) correspondingly and define a type-level prepend operator `::`. This way, using a TList as the type parameter of HList allows for concisely defining a list instance that contains objects of type A, B, and C as `HList[A :: B :: C :: TNil](a,b,c)`. Together with a type-inferring instance-level prepend operator `::`, we can simply write `val x = 123 :: "str" :: HNil` and the type of x will automatically be inferred as `HList[Int :: String :: TNil]`.

Based on such an HList Scala implementation, we defined a heterogeneous term type `TupleTerm` which wraps an HList and thus can contain subterms of different types. A *constructor term* is a specialization of a tuple term that additionally contains a constructor tag, i.e., a class type. Consequently, class `CtorTerm` has two type parameters: the corresponding class type C, and TL, the typelist of its inner HList. Domain objects can now be converted back and forth implicitly as long as pairs of appropriate implicit conversions are provided. The effect is that a domain object can be passed

<sup>1</sup>e.g., J. Nordenberg’s: <http://jnordenberg.blogspot.com/2009/09/type-lists-and-heterogeneously-typed.html>

to any function that expects a term (and vice versa), without having to trigger the conversion explicitly. The implicit conversion definitions can be generated automatically by analyzing the involved EMF metamodels. We provide a Scala script as well as an IDE plug-in together with our iDSL that generates implicit conversions from a given metamodel.

The following listing shows the definition of such an HList-wrapping constructor term type as well as (simplified) definitions of the two implicit conversion functions that are needed to implicitly convert a `ContactInfo` domain object containing a number and a string to a correspondingly type-parameterized `CtorTerm` object and vice versa.

```
class CtorTerm[C, TL <: TList](c: Class[C], subterms: HList[TL])
// domain class ContactInfo and its two implicit conversions:
class ContactInfo(phone: Int, url: String)
implicit def ci2term(ci: ContactInfo): // ContactInfo to Term
  CtorTerm[ContactInfo, ValueTerm[Int] :: ValueTerm[String] :: TNil]
  = CtorTerm(classOf[ContactInfo], ci.phone :: ci.url :: HNil)
implicit def term2ci(t: CtorTerm[ContactInfo, ValueTerm[Int] ::
  ValueTerm[String] :: TNil) = new ContactInfo(t.nth(_0), t.nth(_1))
```

Fig. 2 visualizes how – at runtime – a `ContactInfo` object (from the example in Fig. 1) is converted to a corresponding constructor term object (omitting that values are actually converted to value terms, too).

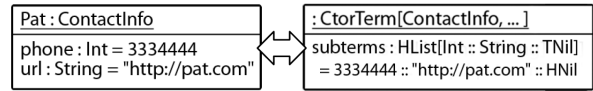


Figure 2: Converting between objects and terms

### 4.2 From Models to Trees with References

Besides the containment tree, an EMF model can contain non-containment references which have to be represented in a corresponding term tree. Therefore, for converting between models and term trees, we traverse the containment hierarchy of the model, create a constructor term for every model element, and keep a trace of every conversion. Then, whenever during traversal we encounter a non-containment reference (which can be easily checked in EMF models), we create an *unresolved* reference term that holds, for now, a reference to the model element that the non-containment reference is pointing to (not the corresponding constructor term). We then add the reference term to a list of unresolved reference terms, and after traversal, we iterate over the list and – using the implicit conversion traces we recorded – we look up which constructor term has been created from which model element, and set the reference in each reference term accordingly. We refer to this process as *resolving references*. In the other direction however, i.e., when creating models from term trees with non-containment references, resolving references is more tricky: When creating domain objects, we cannot pass a reference term which later gets resolved. We solve this as follows: Whenever a non-containment reference is expected, we call a helper function that defers setting the reference and returns null instead. To this function, we pass the referenced constructor term and a pointer to the setter method of the non-containment reference attribute of the created domain object. The null-returning helper function creates a *deferred reference* object which holds the referenced term and the setter method, and adds it to a list of deferred references. After tree traversal, when all domain objects have been created, we resolve references by iterating this list of deferred references and again use traces to find out what domain object has been created from what con-

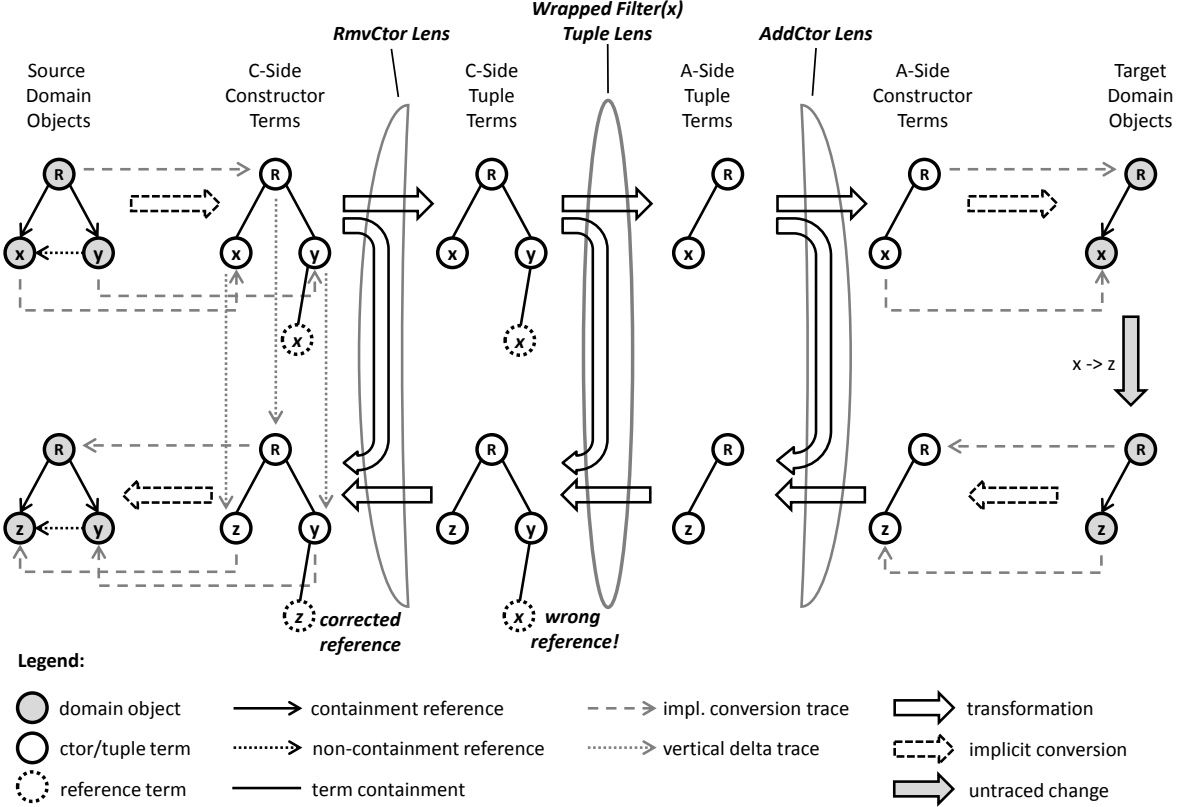


Figure 3: Implicit model to term tree conversion & reference handling with vertical traces

structor term, and use the saved setter methods to replace the nulls in the domain objects with the correct references.

### 4.3 Referential Integrity with Vertical Traces

The strategy to convert between models and term trees with references that we presented so far works well as long as in the forward (abstracting) direction of a lens either no references are abstracted away or as long as they are discarded together with the referenced model elements. If however, the *get* function of a lens discards a reference but keeps the referenced element, this element might be changed on the abstract view side (A-side) which leads to referential corruption when propagating the change back to the concrete source side (C-side): normally, the *put* function of an element-discarding lens (e.g., *filter*) restores the discarded elements by looking them up in the original C-side tree, so it will restore the original references from the concrete source model which might refer to elements that have changed or have been deleted. This problem can be solved when we keep track of what happened to updated model elements on the source side, i.e., when we have a trace of a model element before it is passed to *get* and after it is returned from *put*. Because we implement state-based lenses (and not delta-based lenses [2]), we have no vertical A-side traces which we could translate to such vertical C-side traces.

However, because the asymmetric state-based lens framework defines an incremental binary *put* function which also takes the original C-side model as an additional input, we can create at least C-side vertical traces in the *put* function of a lens: we just have to connect the original C-side model element that is passed as an argument to *put* with

the updated source side element that is created by *put*, before returning it. However, we only have to keep vertical traces of constructor terms, because only their corresponding model elements can actually be referenced. We do not need to keep traces of what happened to potential intermediate structures which have no corresponding model elements. Therefore, we use the following approach: we wrap every lens that translates between constructor terms into a bracket of two semantically transparent helper lenses: the C-side helper lens *RmvCtor* removes the constructor tag of a constructor term (i.e., yielding a tuple term) in the *get* direction (and re-establishes it in the *put* direction) and takes care of the vertical traces, whereas the A-side helper lens *AddCtor* adds a constructor tag in *get* direction (and removes it in the *put* direction). This way, the wrapped lens does not need to know anything about constructor terms and therefore focus on encoding the transformation logic. Furthermore, by wrapping a lens, we mark it as 'finished', i.e., we separate detailed transformation of intermediate structures from 'translation rules' between model elements. This also helps when mixing our bx iDSL with other ways to describe transformations, e.g., with our rule-based unidirectional iDSL [4].

Fig. 3 shows how a model (i.e., a graph of domain objects; the containment tree root is marked with R) is implicitly converted to a tree of terms, and how a non-containment reference becomes a reference term. This tree of terms then goes through the forward transformation *get* of a wrapped *filter* lens (parameterized to filter away every child except x). The term tree directly before and after *filter* consists of tuple terms, whereas before *RmvCtor* and after *AddCtor*, the



term tree consists of constructor terms. However, apart from adding or removing constructor tags, the two helper lenses are semantically transparent as they do not change the structure of the tree. As can be seen, the *filter* lens filters away child *y* which contains a non-containment reference to *x*. However, after the resulting term tree is implicitly converted back to a graph of target domain objects, *x* is replaced by *z* on the A-side, and (because of the state-based lens framework) we have no trace of this change. Therefore, the backward transformation *put* of *filter* simply restores the part of the tree that was filtered away with terms of the original C-side tree including the discarded non-containment reference term that points to *x*. Thus, the restored reference term references term *x* which does not exist anymore so that referential integrity is violated. However now, because *RmvCtor* creates vertical C-side traces (finely dotted), we can resolve wrong references after the tree has passed *RmvCtor*'s *put* function by looking up what has become of the referenced term and correct wrong reference terms before the tree is converted back to a source model with correct references.

## 5. A TYPE-SAFE LENS LANGUAGE

Now that we have term types that preserve the static type information of their domain class counterparts, and can convert between models and term trees, we can start to implement a reusable library of pre-defined lenses which are defined against those term types. The goal of this approach is the following: In spite of the lens library being defined independently from actual domain classes, we want to use static type information to check at compile-time whether a lens (that may be composed out of many small lenses) conforms to the structures it is meant to synchronize, i.e., whether the input/output types of the lens functions match types in the source and target metamodel.

### 5.1 Type-Parameterized Lenses

With a lens that is not parameterized with an edge label – like the *hoist* lens which always performs the same structural modification: lifting a single child out of a tuple term – the two types *C* and *A*, between which a lens translates, only depend on each other: *hoist*'s *C* is always a term with one single edge at the root (this is the C-side constraint of the *hoist* lens) and *A* is always the type of the single child that this edge refers to. Thus, the typelist of term type *C* is a list of length 1 with type *A* as the only component at position 0, written as *A* :: *TNil*. Type *C* can be described as *TupleTerm*[*A* :: *TNil*]. Thus, the type of the *hoist* lens is *Lens*[*TupleTerm*[*A* :: *TNil*], *A*] extending the generic lens type *Lens*[*C* <: *Term*, *A* <: *Term*]. So the only free type-variable of *hoist* is *A*. The following listing shows the complete Scala definition of a type-safe *hoist* lens. Now, when class *Hoist* is type-parameterized with a specific term type, calls to *Hoist*'s lens functions are statically type-checked.

```
class Hoist[A <: Term]() extends Lens[TupleTerm[A :: TNil], A] {
  type C = TupleTerm[A :: TNil] // constrains terms to this shape
  def get(c: C): A = c.subterms.head // simply returns only child
  def put(a: A, c: C): C = this.create(a) // oblivious: put=create
  def create(a: A) = TupleTerm(a :: HNil) // adds edge '_0 -> a'
}
```

As the tree lenses that we are implementing primarily use edge labels (or sets of them) as parameters, and as edge labels in our term data model are translated to indices, we need to encode indices, i.e., natural numbers, as Scala types.

Such type-level numbers can be implemented as *Peano numbers*, i.e., as recursively nested successors of a bottom type which in this case obviously represents the number 0. In a Scala implementation of such type-level numbers, we can define a supertype *Nat*, from which all number types have to inherit, together with type-level number literals like *type \_1*, *type \_2* etc. and corresponding instance-level literals that allow for type-inference. With these number types and number literals, we can define type-safe methods of *HList*, e.g., a type-safe indexed accessor called *nth* by defining a type function *Nth*[*N* <: *Nat*] <: *Term* of *TList*. This type function is used by *HList*'s *nth*-method to determine the result type of accessing the *nth* element of the list. Now, our tuple term class exposes the type function *Nth* of its typelist and the *nth* method of its inner heterogeneous list of subterms.

With this framework of implicit conversions, term types, number types, and type-safe operations on *HLists*, we can define more interesting, parameterized lenses. As an example, we define an atomic *filter* lens that is parameterized with a single index. To distinguish it from the original tree lens which takes a set of labels, we call our variation *FilterN* as it takes a single index *n*. In the get direction, all direct children except the specified one are filtered away, so *FilterN.get* returns a tuple term with a single child. The following listing shows the complete definition of *FilterN* and shows how the *focus* lens can be defined by sequentially composing *FilterN* with the *Hoist* lens we defined earlier.

```
1 class FilterN[N <: Nat, C <: Term](n:N, d:C)
2 extends Lens[C, TupleTerm[C#Nth[N] :: TNil]] {
3   type A = TupleTerm[C#Nth[N] :: TNil]
4   def get(c: C): A = TupleTerm(c.nth(n) :: HNil)
5   def put(a: A, c: C): C = c.replace(n, a.nth(_))
6   def create(a: A) = d.replace(n, a.nth(_)) // using default d
7 }
8 // composing a focus lens using the sequential composition lens:
9 def Focus[N <: Nat, C <: Term](n: N, d: C)
10 = Comp( FilterN(n, d), Hoist[C#Nth[N]]() )
```

*FilterN* has two type parameters: the number type *N* for the specified index, and type *C* of the concrete term. Type *A* does not need to be specified because in this lens, *A* is determined by *C*: *A* is a tuple term with *C*'s *nth* subterm type as the type of the only child. This type is expressed by the help of the *Nth* type function we introduced previously. Thus, the type of *FilterN* is *Lens*[*C*, *TupleTerm*[*C#Nth*[*N*] :: *TNil*]] (line 2). *FilterN* expects two instance-level parameter: index parameter *n* and a default C-side term *d*. From these instance-level parameters, the type-parameters can be inferred. Now, when composing the *focus* lens (line 9), note that the inferred type *A* of *FilterN* has to match type *C* of *Hoist* in order to satisfy the typing constraint of the *Comp* lens. This way, also lens composition is completely type-safe. However, here the type parameter *A* of *Hoist* still has to be specified explicitly in the composition (line 10) which is a problem when composing more complex lenses.

### 5.2 Type-Infering Lens Combinators

Sometimes explicit type parameterization can be avoided by inferring the type from a passed default term. However, often we cannot use domain objects to infer the type from: with lenses that process intermediate terms which have no corresponding domain class, the term type still needs to be specified explicitly which is tedious and error-prone. Imagine a lens that extracts several pieces of information from a source model, and then subsequent lenses rearrange these pieces so that their structure finally matches types of the

target domain. The subsequent lenses need to be parameterized explicitly with the potentially complicated term type which is the output of the first information-extracting lens. To help with this issue, we provide type-inferring lens combinators in our lens iDSL: Most importantly, a type-inferring operator for sequential composition allows for only type-parameterizing the first lens in a chain of lenses explicitly, and let the rest of the chain be parameterized automatically by type inference. We implemented this operator as a right-binding instance- and class-method named `&.`. This way, in a composed lens `l = lens1 &. lens2 &. lens3`, only lens `lens1` needs to be type-parameterized explicitly: the statement is desugared to `l = lens3 . &. (lens2 . &. (lens1))`, where each call of the `&.`-method infers type `A` of the passed lens and creates a correctly typed sequential composition. Furthermore, in order to make the still needed explicit typing of the first lens more comfortable, we provide an operator `$(T)` that is parameterized with a domain type, and infers the (possibly complicated) type of the corresponding constructor term by injecting an appropriate implicit conversion function and inspecting its signature (all at compile-time). This way, one does rarely need to deal with typelists and term types when composing lenses with our iDSL. For also reducing explicit type-parameterizing in parallel lens composition, we provide *lens lists*: Similarly to HLists, there is an end-of-list type called `LLNil` (lens-list-nil) and a type-inferring prepend operator `::`, so that a lens list can be specified as `l1list = lens1 :: lens2 :: lens3 :: LLNil`. The resulting lens list maintains types `C` and `A` of each lens and can then be used, for instance, to parameterize the *WMap* lens combinator which results in a lens that applies a different lens to each subterm of a given tuple term. The *WMap* lens can infer the types of the lens list and therefore also does not need to be type-parameterized explicitly.

### 5.3 Special Lenses for Typed Terms

So far we only presented lenses that were already defined in the original tree lens library (except the two semantically transparent wrapper lenses). Because models which have a containment hierarchy can be converted to term trees with reference terms, these existing tree lenses can be used for describing model transformations. However, because of the different data model of our Scala-based lenses, a few new lenses can be defined. E.g., an important difference from the edge-labeled tree data model of the original state-based tree lenses is that our tree nodes have a type annotation. Therefore, we can describe lenses where this type-annotation determines the behaviour. For instance, we can define a *filter* lens that, instead of a label (i.e., an index), is parameterized with a type. Such a *FilterByType* lens can, for instance, filter for all Integer fields of a model element. Of course, in contrast to filtering for an index which is by definition unique, the same type-annotation can occur multiple times in one term. Therefore, type `C` of this lens is a heterogeneously typed tuple term and type `A` is a homogeneously typed list term; thus, the lens type is `Lens[TupleTerm[TL], ListTerm[T]]`. The semantics of this lens is actually not different from that of the original tree filter lens because the type-annotation is simply an alternative choice of what a label in the original data model can be translated to in our data model.

## 6. FAMILY2PERSONS BIDIRECTIONALLY

In this section we demonstrate the usage of our iDSL by

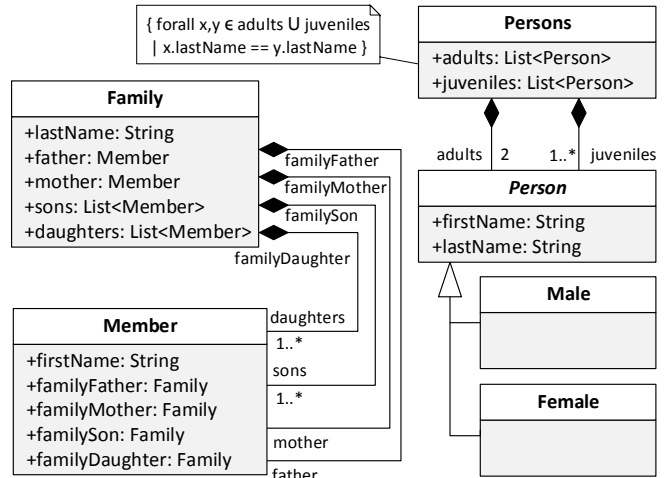


Figure 4: Family2Persons metamodels (bx-version)

presenting a bx version of the Families2Persons<sup>2</sup> example. The modified metamodels are shown in Fig. 4. The Family metamodel stays largely untouched: A family object (the root) contains a last name, two member fields (father and mother), and two list-of-member fields (sons and daughters). A Member object contains the member’s first name and four back-references (i.e., non-containment references) to the family the member belongs to. Note that of those four back-references, three are always null-references, and only that reference which matches the role of the member in the family is set. In the unidirectional version of the example, it is checked which reference is not null to determine the gender of a member. In the Persons metamodel, we added a Persons class which is the root of the containment hierarchy and contains two lists of persons: adults and juveniles. Without the Persons class, the Persons metamodel would not fulfill our requirement that every model must have a containment root object. The distinction between adults and juveniles allows us to implement the example in a state-based fashion (i.e., without horizontal inter-model traces) and without having to deal with heuristics-based name matching etc. which would distract from the actual synchronization logic. Also, in the Person class, first name and last name are two separate fields instead of one full name field to avoid cluttering the example with string analysis specifics. Furthermore, and importantly, we added an equality constraint that says that every person in a Persons object has to have the same last name. This constraint is restrictive and might seem to render the synchronization example trivial but it cannot be avoided when trying to stay close to the original unidirectional example, i.e., when describing the transformation in the direction from Family to Persons: In asymmetric lenses the forward direction is the abstracting one; thus, a persons model cannot contain multiple last names because otherwise it would not be fully determined by a C-side family model and therefore no abstraction<sup>3</sup>. However, one can change all last names in a persons model and propagate this change back to the family model. Also adding and deleting children is a supported A-side modification. In order to show how a composed family2persons lens works,

<sup>2</sup>[http://wiki.eclipse.org/ATL/Tutorials\\_-\\_Create\\_a\\_simple\\_ATL\\_transformation](http://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation)

<sup>3</sup>one could imagine a persons model as the result of a last-name-query to a bigger persons database to render the bx version more useful

we demonstrate how a term that represents a family model is stepwise rewritten so that it finally has a structure that matches a persons model. Because list handling is a bit involved, we omit the children lists in this demonstration, and suppose that a family only has a father and a mother, i.e., a family term only consists of three subterms: a value term of type String for the last name and two constructor terms of type Member. Correspondingly, suppose for now that a persons term only consists of two subterms: a constructor term of type Male and a constructor term of type Female. In the following sequence of term rewritings, we denote a tuple term by  $(subterm1, subterm2, \dots)$  with an optional constructor prefix, and a string value term by  $'value'$ . We denote a null-valued non-containment reference term by  $\emptyset$ , and a non-null non-containment reference term by  $\mapsto$ . Next to the current term structure, we show the (parameterized) lens whose forward transformation  $get$  (denoted by  $\nearrow$ ) is applied to yield the next term in the rewriting sequence, i.e., the term rewriting rule that is applied to that term. Remember that the  $WMap$  lens is parameterized with as many lenses as the number of subterms of the tuple term it is applied to, and then applies each lens to one subterm. For brevity, we start the term rewriting with all constructor tags already removed.

```

('Simpson', ('Homer',  $\mapsto$ ,  $\emptyset$ ,  $\emptyset$ ,  $\emptyset$ ), ('Marge',  $\emptyset$ ,  $\mapsto$ ,  $\emptyset$ ,  $\emptyset$ ))
  ( $\nearrow$ WMAP(ID, FOCUS(0), FOCUS(0)))
('Simpson', 'Homer', 'Marge')      ( $\nearrow$ DUPLICATE(0))
('Simpson', 'Simpson', 'Homer', 'Marge')  ( $\nearrow$ SPLIT(2))
(('Simpson', 'Simpson'), ('Homer', 'Marge'))
  ( $\nearrow$ REVERSE)
(('Homer', 'Marge'), ('Simpson', 'Simpson'))  ( $\nearrow$ ZIP)
(('Homer', 'Simpson'), ('Marge', 'Simpson'))
  ( $\nearrow$ WMAP(ADDCTOR(Male), ADDCTOR(Female)))
(Male('Homer', 'Simpson'), Female('Marge', 'Simpson'))
  ( $\nearrow$ ADDCTOR(Persons))

```

Next, we show how a complete family2persons lens can be constructed, including all list handling and constructor handling. First, we use an idealized syntax of our lens DSL and denote sequential lens composition with  $\&$ .

```

adultName = RMVCTOR(Member) & FOCUS(0)
childNames = LISTMAP(SPLIT(1) & REVERSE) & FACTORIZE & FOCUS(1) & LISTMAP(HOIST)
distribute = SPLIT(1) & TUPLEDISTRIBUTE & WMAP(ID, ID, DISTRIBUTE, DISTRIBUTE)
reverse = WMAP(REVERSE, REVERSE, LISTMAP(REVERSE), LISTMAP(REVERSE))
addCtors = WMAP(ADDCTOR(Male), ADDCTOR(Female), LISTMAP(ADDCTOR(Male), LISTMAP(ADDCTOR(Female)))
sort = SPLIT(2) & MAP(SUPERTYPELISTCONCAT(Person, Male, Female))
families2persons = WMAP(ID, adultName, adultName, childNames, childNames) & distribute & reverse & addCtors & sort & ADDCTOR(Persons)

```

The presented composition contains a few lenses which were not defined in the original tree lenses. E.g., the *Distribute* and *Factorize* lenses mimic the application of the distributive property from basic algebra. Because they du-

plicate values (in either the one or the other direction), they rely on equality constraints in the involved metamodels. *SupertypeListConcat* is a special lens that works with type-annotations: it concatenates two lists of different types to one list of a common supertype. In the backwards direction, it splits a list depending on the specific subtypes of the elements. The type of *SupertypeListConcat* is  $\text{Lens}[\text{TupleTerm}[\text{ListTerm}[\text{SUB1}], \text{ListTerm}[\text{SUB2}], \text{ListTerm}[\text{SUP}]]]$ , where  $\text{SUP} <: \text{Term}$ ,  $\text{SUB1} <: \text{SUP}$ , and  $\text{SUB2} <: \text{SUP}$ .

Now, let us see how the lens construction looks in our Scala iDSL. The following listing shows a very similar lens definition as the one before (only decomposed slightly differently). Obviously, type annotations make the description in our iDSL more noisy than the clean description in the idealized syntax before<sup>4</sup>. We could easily achieve a similarly clean iDSL syntax in Scala, however, not while at the same time getting automatic (and extensive) static type-checking. Now, as one can imagine, when constructing a lens as complex as this one (or even much more complex), automatic static type-checking can be tremendously helpful, as there are plenty of possibilities to make mistakes when composing many small lens combinators. Because we keep track of most types, most of such mistakes are detected automatically at compile-time and highlighted with standard Scala tooling. Note that in our iDSL we provide the language construct `wrap(...)` as `[SourceType, TargetType]` to wrap a tuple lens in between a *RmvCtor* and *AddCtor* lens (to be precise: if type A is a value term, only *RmvCtor* is used).

```

1 // constructing a type-safe families2persons lens:
2 val adultName = wrap( Focus(_0, Member("")) ) as[Member,String]
3
4 val childNames = wrap( ListMap(Split(_1, $[Member]) &: Reverse)
5   &: Factorize &: Focus(_1,Term(Term(NullRef::NullRef::NullRef::
6     NullRef::HNil)::List("")::HNil)) as[List[Member],List[String]]
7
8 val distribute1 = Split(_1, $[Family]) &: TupleDistribute
9
10 val distribute2 = WMap(Id[String]::Id[String]::
11   Distribute[String,String]::Distribute[String,String]::LLNil)
12
13 val strrev = Reverse[String]::String::TNil]
14
15 val reverse = WMap(strrev :: strrev :: ListMap(strrev) ::
16   ListMap(strrev) :: LLNil)
17
18 val addCtors = WMap(AddCtor($[Male]) :: AddCtor($[Female]) ::
19   ListMap(AddCtor($[Male])::ListMap(AddCtor($[Female])::LLNil)
20   &: Split(_2)
21
22 val sort = Map(SupertypeListConcat($[Person], $[Male], $[Female]))
23
24 val extractNames = WMap( Id[String] :: adultName :: adultName ::
25   childNames :: childNames :: LLNil)
26
27 val rearrange = extractNames &: distribute1 &: distribute2 &:
28   addCtors &: sort
29
30 val families2persons = wrap(rearrange) as[Family,Persons]

```

The above is valid Scala code and fully type-checked. Note that, by using type-inferring operators, only a few lenses need to be typed explicitly. The final lens can directly be used to synchronize family and persons models which will be automatically converted to (and from) corresponding term trees. The availability of all required (possibly generated) implicit conversions is checked automatically at compile-time when wrapping the lens.

<sup>4</sup>Also, the need to provide a default term for the unary *create* function is sometimes distracting (e.g., in line 5-6 for the focus lens). If we did not allow for initialization from the abstract side – i.e., define a lens only as a tuple of *get* and *put* – lens descriptions would look cleaner.



## 7. RELATED WORK & CONCLUSIONS

To the best of our knowledge, we are the first to present a bx language for *model* transformations as an iDSL in a statically typed JVM-language. Originally, our approach to embed a compositional, term-rewriting-based language as an iDSL in Scala was inspired by the work of Sloane [9], who implemented the unidirectional term-rewriting language *Stratego* as an iDSL in Scala. However, this iDSL allows for little static verification because Scala's type system is not used to the same extent as in our approach. Cuadrado et al. presented RubyTL [1], a unidirectional transformation language implemented as an iDSL in Ruby. However, because Ruby is dynamically typed and is no JVM-language, possibilities for static verification are very limited. Therefore, we presented a similar, ATL-inspired transformation language implemented in Scala that allows for more EMF-integration, tool-support, and static verification [4]. We think that for bx, static verification and tool-support are even more important. Regarding statically type-checked bx, Pacheco & Cunha presented a tree lenses iDSL in Haskell [7]. However, besides providing no JVM-integration this way, their work also does not aim for adapting tree lenses to model transformations.

Regarding bx languages for model transformations, there are many promising approaches but, as far as we know, none has been implemented as an iDSL, yet. They can be roughly divided into asymmetric, symmetric, and bijective approaches. For the asymmetric case, *GRoundTram*, developed by Hidaka et al. [5], is one of the most mature bx tools which provides a graph-query language called *UnQL+* for specifying asymmetric bx. Such a query language is a clear advantage over our lens iDSL in terms of usability, because it allows for defining graph-traversals relatively comfortably (which is cumbersome with our root-oriented combinator approach). This is partly due to the fact that *GRoundTram* is from the ground up graph-based – and not as our approach essentially tree-based – but also partly because it provides less static type analysis which makes graph traversals easier. There are attempts to integrate *GRoundTram* with EMF and with ATL, but until now both is limited and not seamless.

For the symmetric case, there are the QVT standard, with its *QVT-Relations* bx language, and *Triple Graph Grammars* (TGG) by Schürr et al. [8]. Both are rule-based approaches. However, QVT-R has semantic issues concerning non-bijective bx [10] which might be the reason why there is no QVT-R tool anymore which is actively developed. TGGs have a solid semantic foundation. However, TGG-based tools that support bx and integrate seamless with EMF only emerged recently. Because TGGs are also graph-based, they do not require an underlying spanning containment tree, and are in general more expressive concerning changes of non-containment references. Furthermore, there are *delta-based lenses* [2] which can be either symmetric or asymmetric. Because delta-based lenses separate update-alignment from update-propagation, they can synchronize graph-based models as long as a correct alignment (i.e., vertical traces) of the involved models can be provided.

Many of the aforementioned bx approaches are more powerful or allow for synchronizations to be described more comfortably. However, because none of these approaches is implemented as an iDSL in a JVM-based GPL, none of them is as tool-independent as our approach: transformation description with intelliSense-like code-completion, transformation execution, debugging, and technological integration can

all be provided by any of several available Scala IDE plugins. Therefore, one does not rely on the ongoing development and maintenance of bx tooling. All that is needed, is to install a Scala tool-set and import the iDSL library in an existing EMF- or Java-based project. Furthermore, as far as we know, with none of the presented approaches, it is possible to mix bx both with unidirectional transformations (e.g., using our iDSL from [4]) and with GPL-coded transformations. In practice, this can be an important advantage concerning developer acceptance because it allows for gradual migration from unidirectional transformation descriptions to bx: Developers who do not immediately see how to solve a synchronization task using a special transformation language can first use Java or Scala as a GPL and can later gradually migrate to a bx implementation for reducing the long-term maintenance overhead of pairs of unidirectional transformations or GPL-coded synchronizations.

However, because the advantages of our approach mainly stem from the Scala-based iDSL approach, we rather want to promote the general approach of implementing bx languages as iDSLs in Scala than the specific state-based tree-lens iDSL that we presented. Its implementation allowed us to demonstrate how much expressiveness and static analysis can be achieved by implementing a bx language in Scala. Scala's type system is capable of unrestricted compile-time recursion which allowed for extensive static guarantees even for complicated lenses. Therefore, it would be highly interesting to apply the approach to other bx languages, e.g., *GRoundTram*, delta-based lenses, or TGGs. Concerning the latter, we already showed that the implicit conversion mechanism is particularly suited for implementing rule-based iDSLs [4].

## Acknowledgements

We like to thank the anonymous reviewers for comments on a preliminary version of this paper. This work was supported by the BMBF, FHprofUnt grant 17075A10 (MOSES).

## 8. REFERENCES

- [1] J. Cuadrado, J. Molina, and M. Tortosa. RubyTL: A Practical, Extensible Transformation Language. In *MDA - Foundations and Applications*, pages 158–172. Springer, 2006.
- [2] Z. Diskin, Y. Xiong, and K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *Journal of Object Technology*, 10:6: 1–25, 2011.
- [3] J. N. Foster, M. Greenwald, J. Moore, B. Pierce, and A. Schmitt. Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-Update Problem. *ACM Trans. Program. Lang. Syst.*, 29(3), 2007.
- [4] L. George, A. Wider, and M. Scheidgen. Type-Safe Model Transformation Languages as Internal DSLs in Scala. In *Int'l Conf. on Model Transformation (ICMT'12)*, Prague, volume 7307 of *LNCIS*, pages 160–175. Springer, 2012.
- [5] S. Hidaka, Z. Hu, K. Inaba, H. Kato, and K. Nakano. *GRoundTram*: An Integrated Framework for Developing Well-behaved Bidirectional Model Transformations. In *Int'l Conf. on Automated Software Engineering (ASE 2011)*, Oread, Kansas, USA, pages 480–483. IEEE, 2011.
- [6] O. Kiselyov, R. Lämmel, and K. Schupke. Strongly Typed Heterogeneous Collections. In *Haskell '04: ACM SIGPLAN Workshop on Haskell*, pages 96–107. ACM, 2004.
- [7] H. Pacheco and A. Cunha. Generic Point-Free Lenses. In *10th Int'l Conf. on Mathematics of Program Construction (MPC'10)*, *LNCIS* 6120, pages 331–352. Springer, 2010.
- [8] A. Schürr and F. Klar. 15 Years of Triple Graph Grammars. In *ICGT*, pages 411–425, 2008.
- [9] A. M. Sloane. Experiences with Domain-Specific Language Embedding in Scala. In *Int'l Workshop on Domain-Specific Program Development*, 2008.
- [10] P. Stevens. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. *Software and Systems Modeling*, 9(1):7–20, 2010.

# Towards a Framework for Multidirectional Model Transformations

Nuno Macedo  
HASLab  
INESC TEC & Universidade  
do Minho, Braga, Portugal  
nfmacedo@di.uminho.pt

Alcino Cunha  
HASLab  
INESC TEC & Universidade  
do Minho, Braga, Portugal  
alcino@di.uminho.pt

Hugo Pacheco  
National Institute of  
Informatics  
Tokyo, Japan  
hpacheco@nii.ac.jp

## ABSTRACT

The *Query/View/Transformation Relations* (QVT-R) standard for bidirectional model transformation is notorious for its underspecified semantics. When restricted to transformations between pairs of models, most of the ambiguities and omissions have been addressed in recent work. Nevertheless, the application of the QVT-R language is not restricted to that scenario, and similar issues remain unexplored for the multidirectional case (maintaining consistency between more than two models), that has been overlooked so far.

In this paper we first discuss ambiguities and omissions in the QVT-R standard concerning the multidirectional transformation scenario, and then propose a simple extension and formalization of the checking and enforcement semantics that clarifies some of them. We also discuss how such proposal could be implemented in our **Echo** bidirectional model transformation tool. Ours is just a small step towards making QVT-R a viable language for bidirectional transformation in realistic applications, and a considerable amount of basic research is still needed to fully accomplish that goal.

## Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques

## 1. INTRODUCTION

In model-driven engineering (MDE), models are the main development artifact. Typically, multiple models may coexist in the same environment, to represent different views of the overall system or similar components at different levels of abstraction, and all these models must ideally be kept consistent with each other. In the past years, extensive work on *bidirectional model transformations* [3] has been devoted to the particular purpose of maintaining the consistency of two models. One of the most popular approaches in MDE is the OMG's QVT standard [9], and in particular the QVT-R language, that proposes describing a bidirectional transformation as a declarative relation between two meta-models.



Figure 1: Configuration (CF) and feature model (FM).

The standard prescribes two modes to interpret a QVT-R specification: *checkonly mode* tests the consistency between particular models and *enforce mode* runs a transformation in a particular direction to repair inconsistent models. Thus, a QVT-R transformation between two meta-models can be understood in the abstract framework of *constraint maintainers* [8]: a specification denotes a consistency relation  $R \subseteq A \times B$ , from which a forward transformation  $\vec{R} : A \times B \rightarrow B$  and a backward transformation  $\overleftarrow{R} : A \times B \rightarrow A$ , that modify one of the elements to restore consistency, are inferred. In this paper we will be dealing with multiple target models, so these will instead be denoted by  $\vec{R}_A : B \rightarrow A$  and  $\overleftarrow{R}_B : A \rightarrow B$ , the subscript identifier denoting the direction of the transformation and the fact that it retrieves information from the original target models.

Acceptance and development of effective tool support for the QVT-R standard has been slow, possibly due to ambiguities in its checking and enforcement semantics. For the bidirectional scenario (maintaining consistency between two models), most of the issues have been clarified in recent work [10, 1] and implementations with precise semantics already exist, namely our own **Echo** tool [6]. Nonetheless, the bidirectional scenario is not sufficient to tackle some applications. An arbitrary number of models may coexist in the same model-driven environment, and their complex interrelationship may not be decomposable into a set of bidirectional relationships to be maintained separately.

As an example, consider the problem of keeping the consistency between a *feature model* and a set of valid *configurations*. For the sake of simplicity, assume that feature models FM consist of named features, that may or not be mandatory, and configurations CF are simply a set of selected features; the respective meta-models are depicted in Figure 1.

The relationship  $F \subseteq \text{FM} \times \text{CF}^k$  between a feature model and  $k$  configurations can be decomposed into two parts  $F = MF \cap OF$ : relation  $MF \subseteq \text{FM} \times \text{CF}^k$  expresses that mandatory features match exactly the set of features appearing in every CF; and relation  $OF \subseteq \text{FM} \times \text{CF}^k$  expresses that the FM contains at least the union of all selected features. Note that due to the intention of having features present in all CFs set

as mandatory in the FM, relation  $MF$  cannot be decomposed into  $k$  bidirectional relations between the FM and each CF.

This kind of *multidirectional* scenario is already informally foreseen in the QVT-R standard, that admits an arbitrary number of domains in a QVT-R relation. However, the proposed checking semantics are too inflexible and only able to represent a restricted subset of consistency relations. For instance, none of the above relations can be specified using the standard checking semantics. Moreover, the standard hints at an enforcement semantics with very limited applicability. Namely, from a consistency relation such as  $F$ , the standard only prescribes the derivation of two transformations:

- $\vec{F}_{FM} : CF^k \rightarrow FM$ , for propagating updates to the configurations back to the feature model;
- $\vec{F}_{CF}^i : FM \times CF^{k-1} \rightarrow CF$  for any  $i \in 0..k$ , for propagating updates to the feature model and the remaining configurations into a specific target configuration.

In the multidirectional scenario this is a very restrictive view of the enforcement semantics, as the user may wish to restore consistency in many ways depending on the context, leading to different update propagation transformations. Consider, e.g., the following interesting and alternative instantiations:

- $\vec{F}_{CF^k} : FM \rightarrow CF^k$ . This would allow updates to the feature model to be propagated to more than one configuration. For example, if a feature is changed to mandatory it must be selected in all configurations; this simple update could not be handled by the standard transformations, since full consistency could not be restored by a single update translation.
- $\vec{F}_{FM \times CF^{k-1}}^i : CF \rightarrow FM \times CF^{k-1}$  for any  $i \in 0..k$ . This would provide more flexibility in the propagation of updates to a configuration, as all the remaining artifacts are allowed to change. For example, if name of a feature is changed, the natural way to recover consistency is to change the name of that feature in all the remaining configurations and in the feature model.

The main goal of this paper is to start shedding some light on this currently unexplored multidirectional scenario. In particular, we explore the applicability of QVT-R for specifying multidirectional model transformations and discuss some semantic issues that arise in this setting. To overcome its current limitations, we propose a simple extension that enables the specification of interesting multidirectional transformations, and discuss how to infer different kinds of consistency-restoring transformations from the *same* multidirectional specification. Finally, we show how Echo [7], a tool supporting QVT-R bidirectional transformations, could be easily adapted to accommodate such extensions.

## 2. QVT-R CHECKING SEMANTICS

Typical model transformation languages (like QVT-R [9] and ATL [5]) provide mechanisms that allow reasoning about transformations in a structured way, rather than simply specifying arbitrary constraints over the models in some general constraint language (like OCL). As evidence, both these languages rely on domain patterns, controlling the elements over which a transformation is applied, while QVT-R supports additional pre- and post-conditions. A QVT-R program is defined as a set of *relations* in the following syntax.

```
[top] relation R {
  [variable declarations]
  domain m1 a1 : A1 { π1 }
  ...
  domain mn an : An { πn }
  [when { ψ }] [where { φ }] }
```

In this notation,  $\pi_i$  denotes a domain pattern over an element  $a_i$  of model  $m_i$  (for  $i \in 1..n$ ), and  $\psi$  and  $\phi$  are arbitrary pre- and post-conditions. According to the standard, testing the consistency specified by top relations consists of running  $n$  *directional* tests (denoted by the subscript meta-model identifier), each validating one of the models, as:

$$R(m_1 : M_1, \dots, m_n : M_n) \equiv \bigwedge_{i \in 1..n} R_{M_i}(m_1, \dots, m_n)$$

For each of these  $R_{M_i}$  relations, the idea is that if  $\psi$  holds, then for all  $a_j$  elements such that  $\pi_j$  holds, for  $j \neq i$ , there must exist an element  $a_i$  such that  $\pi_i$  and  $\phi$  hold. For a non-top relation, its constraints must hold only when called by other relations. In the bidirectional case, we end up with:

$$\begin{aligned} R_{M_1}(m_1 : M_1, m_2 : M_2) &\equiv \\ &\forall xs \mid \psi_{M_1} \wedge \pi_{M_2} \Rightarrow (\exists ys \mid \pi_{M_1} \wedge \phi_{M_1}) \\ &\textbf{where } xs = \text{fv}(\psi \wedge \pi_{M_2}), ys = (\text{fv}(\pi_{M_1} \wedge \phi)) - xs \\ R_{M_2}(m_1 : M_1, m_2 : M_2) &\equiv \\ &\forall xs \mid \psi_{M_2} \wedge \pi_{M_1} \Rightarrow (\exists ys \mid \pi_{M_2} \wedge \phi_{M_2}) \\ &\textbf{where } xs = \text{fv}(\psi \wedge \pi_{M_1}), ys = (\text{fv}(\pi_{M_2} \wedge \phi)) - xs \end{aligned}$$

This checking semantics has a close correspondence to the enforcement semantics: roughly, we just need to replace existential quantifiers for generation procedures [9]. Relations may call other relations in their pre- and post-conditions, which are also run in the appropriate direction (hence the identifier on  $\psi$  and  $\phi$  denoting the required direction).

### 2.1 Issues with the Multidirectional Scenario

Back to our running example from Section 1, consider that we have a pair of configurations ( $k = 2$ ). How can the  $MF$  consistency relation be specified in QVT-R? As a first attempt, let us consider the following specification.

```
top relation MF { n : String;
  domain cf1 s1 : Feature { name = n }
  domain cf2 s2 : Feature { name = n }
  domain fm f : Feature { name = n,
    mandatory = true } }
```

The free variable  $n$  relates selected features in each configuration with mandatory features in the feature model, resulting the consistency relation:

$$\begin{aligned} MF(cf_1 : CF_1, cf_2 : CF_2, fm : FM) &\equiv \\ MF_{FM}(cf_1, cf_2, fm) &\wedge \\ MF_{CF_1}(cf_1, cf_2, fm) &\wedge MF_{CF_2}(cf_1, cf_2, fm) \end{aligned}$$

Each of these directional tests is then concretized as:

$$\begin{aligned} MF_{FM}(cf_1 : CF, cf_2 : CF, fm : FM) &\equiv \\ \forall n : \text{String}, s_1 : \text{Feature}_{cf_1}, s_2 : \text{Feature}_{cf_2} & \mid \\ n = s_1.\text{name} \wedge n = s_2.\text{name} &\Rightarrow \\ (\exists f : \text{Feature}_{fm} \mid n = f.\text{name} \wedge f \in \text{mandatory}) & \\ MF_{CF_1}(cf_1 : CF, cf_2 : CF, fm : FM) &\equiv \\ \forall n : \text{String}, f : \text{Feature}_{fm}, s_2 : \text{Feature}_{cf_2} & \mid \\ n = s_1.\text{name} \wedge n = f.\text{name} \wedge f \in \text{mandatory} &\Rightarrow \\ (\exists s_1 : \text{Feature}_{cf_1} \mid n = s_1.\text{name}) & \\ MF_{CF_2}(cf_1 : CF, cf_2 : CF, fm : FM) &\equiv \dots \end{aligned}$$

But let us concretely analyze the meaning of these predicates.  $MF_{FM}$  expresses part of the intended behavior — if the two configurations have the same selected feature then such feature is mandatory. It can be rephrased as:

$$MF_{FM} (cf_1 : CF, cf_2 : CF, fm : FM) \equiv \\ \text{Feature}_{cf_1}.\text{name} \cap \text{Feature}_{cf_2}.\text{name} \subseteq \\ (\text{Feature}_{fm} \cap \text{mandatory}).\text{name}$$

However, the reverse implication

$$MF_{CF_1 \times CF_2} (cf_1 : CF, cf_2 : CF, fm : FM) \equiv \\ (\text{Feature}_{fm} \cap \text{mandatory}).\text{name} \subseteq \\ \text{Feature}_{cf_1}.\text{name} \cap \text{Feature}_{cf_2}.\text{name}$$

is not entailed by  $MF_{CF_1}$  and  $MF_{CF_2}$ . Looking at  $MF_{CF_1}$ , the selection of the  $s_1$  feature depends both on  $f$  and  $s_2$ , thus, if there are *no* selected features in  $cf_2$ ,  $MF_{CF_1}$  will be trivially true due to the empty range in the universal quantification. This problem persists whatever the domain patterns (and pre- or post-conditions), and the intended specification for  $MF$  cannot be realized by any QVT-R relation (with the standard semantics) between features in the three models.

This problem could be easily solved if we could control the extent of the universal quantifications in the semantics of  $MF_{CF_1}$  and  $MF_{CF_2}$ , namely to range only over the feature model and ignore the remaining configurations.

$$MF_{CF_1} (cf_1 : CF, cf_2 : CF, fm : FM) \equiv \\ \forall n : \text{String}, f : \text{Feature}_{fm} \mid \\ n = f.\text{name} \wedge f \in \text{mandatory} \Rightarrow \\ (\exists s_1 : \text{Feature}_{cf_1} \mid n = s_1.\text{name}) \\ MF_{CF_2} (cf_1 : CF, cf_2 : CF, fm : FM) \equiv \\ \forall n : \text{String}, f : \text{Feature}_{fm} \mid \\ n = f.\text{name} \wedge f \in \text{mandatory} \Rightarrow \\ (\exists s_2 : \text{Feature}_{cf_2} \mid n = s_2.\text{name})$$

The conjunction of these predicates entails the missing part of the desired  $MF$  specification, and hints at a possible extension to the standard checking semantics that largely improves its expressiveness, as described in the next section.

Although our example could alternatively be interpreted as a bidirectional transformation between a feature model  $FM$  and a tuple of configurations  $CF^k$ , in general the  $n$  models may be of different nature. Moreover, the standard checking semantics could not be reproduced under such view.

## 2.2 Extending the Standard Semantics

As the previous section makes clear, standard QVT-R relations are not suitable for expressing many transformations of interest, namely those where relationships are not symmetric. In fact, this is already a problem in the bidirectional setting (for example, how to express a plain subset relationship?), but is aggravated in the multidirectional setting due to the explosion of possible dependencies between domains. In this paper, we propose precisely to extend QVT-R with a language of dependencies between domains in order to express the desired directionality of the checking semantics.

Let  $dom R$  denote the set of meta-model identifiers  $M_1, \dots, M_n$  in a relation  $R \subseteq M_1 \times \dots \times M_n$ . A *checking dependency*  $S \rightarrow T$  for  $R$ , where  $S \subseteq dom R$  is a set of identifiers and  $T \in dom R$  a single identifier (with  $T \notin S$ ), states that the model conforming to  $T$  depends on all the models conforming to the meta-models in  $S$ . Formally, the semantics of a rule  $R$  according to a dependency  $S \rightarrow T$ , denoted by  $R_{S \rightarrow T}$ , prescribes that  $R$  should be checked by quantifying

universally over all the domains in  $S$  and, when the respective domain patterns and pre-condition hold, demanding an element satisfying the respective domain pattern and post-condition to exist in the  $T$  domain. The set of checking dependencies attached to a relation  $R$  will be denoted by  $\underline{R}$ . The semantics of a top relation  $R$  is now the conjunction of all directional checks  $\bigwedge_{d \in \underline{R}} R_d (m_1, \dots, m_n)$ .

For example, to obtain the desired specification of the  $MF$  relation, it suffices to attach to the above QVT-R specification the dependencies  $\underline{MF} \equiv \{CF_1 CF_2 \rightarrow FM, FM \rightarrow CF_1, FM \rightarrow CF_2\}$ . This extension is conservative, in the sense that the standard semantics can still be specified by setting:

$$\underline{R} \equiv \bigcup_{i \in \{0..n\}} (dom R \setminus M_i \rightarrow M_i)$$

The  $OF$  relation, stating that the union of selected features should be included in the set of all available features

$$OF_{FM} (cf_1 : CF, cf_2 : CF, fm : FM) \equiv \\ \text{Feature}_{cf_1}.\text{name} \cup \text{Feature}_{cf_2}.\text{name} \subseteq \text{Feature}_{fm}.\text{name}$$

can now be represented by the QVT-R relation

```
top relation OF { n : String;
  domain cf1 s1 : Feature { name = n }
  domain cf2 s2 : Feature { name = n }
  domain fm f : Feature { name = n } }
```

associated with the checking dependencies  $\underline{OF} \equiv \{CF_1 \rightarrow FM, CF_2 \rightarrow FM\}$ . Of course, this extension also improves the expressiveness in the bidirectional setting, allowing for example to specify a subset constraint between two domains by just attaching one dependency between them. Note that, at this point, we are just disregarding the dependencies implied by the QVT-R standard. Expressing our dependency would require some sort of extended QVT-R syntax.

Although at first sight this extension may seem too conservative, the fact is that from these simple dependencies more complex ones can be built. In particular, multiple model dependencies can be attained through the entailment  $\{M_1 \rightarrow M_2, M_1 \rightarrow M_3\} \vdash \{M_1 \rightarrow M_2 M_3\}$  (thus resulting in the expected  $MF_{CF_1 \times CF_2}$ ) while dependencies over unions of models can be attained through  $\{M_1 \rightarrow M_3, M_2 \rightarrow M_3\} \vdash \{M_1 \mid M_2 \rightarrow M_3\}$  (from which  $OF_{FM}$  arises).

## 2.3 Relation Invocations

As in the bidirectional scenario, multidirectional relation calls must preserve the direction of the caller. However, the QVT-R syntax does not guarantee that every relation in a specification can be run in the same direction, e.g., nothing prevents a relation  $R \subseteq CF^k \times FM$  running in the  $FM$  direction from calling another relation  $S \subseteq CF^k$ , which has no  $FM$  direction. The standard is omissive about these situations. The newly introduced checking dependencies must also be taken into consideration, e.g., should a relation  $\underline{R} \equiv \{M_1 \rightarrow M_2\}$  be allowed to call another relation  $\underline{S} \equiv \{M_2 \rightarrow M_1\}$ ? We think the answer should be no, and this situation should be flagged as a typing error at static time.

Notwithstanding, it is worth noting that the dependencies of  $R$  and  $S$  need not be perfect matches. In fact, a relation  $\underline{R} \equiv D$  may be called in the direction  $R_d$  by another relation  $\underline{S} \equiv \{\dots d \dots\}$  if  $D \vdash d$ , i.e.,  $D$  entails  $d$ . In our restricted language this will allow, for instance, the call  $R_{M_1 \rightarrow M_3}$  when  $\underline{R} \equiv \{M_1 \rightarrow M_2, M_2 \rightarrow M_3\}$ , since  $\{M_1 \rightarrow M_2, M_2 \rightarrow M_3\} \vdash M_1 \rightarrow M_3$ . Since our dependencies are equivalent to *Horn clauses* (disjunctions with a single positive literal) this “type checking” can be done in linear time.

### 3. QVT-R ENFORCEMENT SEMANTICS

In [6], we proposed a technique for bidirectional QVT-R model transformation following the least-change principle [8], which was implemented in the bidirectional transformation tool `Echo` [7]. Given a binary consistency relation  $R \subseteq M_1 \times M_2$  and a model distance metric  $\Delta_{M_1} : M_1 \times M_1 \rightarrow \mathbb{N}$ , if  $m_1$  and  $m_2$  are two inconsistent models, the new model  $m'_1$  produced by transformation  $\vec{R}_{M_1}$  is a consistent model that is as close as possible to the original one, according to the given metric. This results in a clear and predictable enforcement semantics. Note that although  $\vec{R}_{M_1}$  is a transformation from  $M_2$  to  $M_1$ , the original model  $m_1$  is also taken into consideration in order to achieve minimality. The technique embeds the QVT-R checking semantics in `Alloy` specifications [4], then calling its model finder in an iterative process of searching for all consistent models at increasing distance from the original  $m_1$  (or alternatively, using optimizing solvers, such as PMax-Sat, as proposed in a recent extension [2]). The concretization of the  $\Delta$  metric is outside the scope of this paper, and the reader is redirected to the original paper. This transformation technique requires only the definition of a consistency relation and a suitable distance metric for the target domain, thus extending it for the multidirectional scenario requires only the definition of suitable distances for the selected output.

Let us use the sample transformations from Section 1 to explore the transformation space. Those with a single output model can be trivially applied. For instance, assuming a model distance  $\Delta_{FM}$ ,  $\vec{F}_{FM} : CF^k \rightarrow FM$  would produce a feature model  $fm'$  consistent with the input configurations ( $MF (cf_1, \dots, cf_k, fm')$ ) and closest to the original feature model (minimizing  $\Delta_{FM}(fm, fm')$ ). Similarly for  $\vec{F}_{CF}^i$ . As for the tuple returning transformations, a naive way to achieve the combined distance of the target models is to add up the distance between every model, e.g.,

$$\Delta_{CF^k}((cf_1, \dots, cf_k), (cf'_1, \dots, cf'_k)) = \Delta_{CF}(cf_1, cf'_1) + \dots + \Delta_{CF}(cf_k, cf'_k)$$

for the transformation  $\vec{F}_{CF^k} : FM \rightarrow CF^k$  that updates all configurations. Of course, this means that all changes in all the models have the same weight, what may not be desirable (e.g., in  $\vec{F}_{FM \times CF}^i : CF \rightarrow FM \times CF^{k-1}$  changes to configurations could be prioritized over those to feature models). We leave that customization for future work.

When applying a transformation, the user must be aware that not all update directions are able to restore the consistency of the system. Consider, for instance, that a new mandatory feature is introduced in the FM model. Then  $\vec{F}_{CF}^i$ , which updates a single model, will clearly not be able to restore consistency of the model-driven environment. Instead, the user should apply  $\vec{F}_{CF^k}$  and update all CFs.

### 4. FUTURE WORK

To the best of our knowledge, there exists no work dedicated to multidirectional transformations in QVT-R (or in any other model transformation language). Therefore, the natural direction for this work is to collect reasonable case studies and to study syntactic means to describe multidirectional transformations in order to validate our approach.

We have shown that the checking semantics proposed in

the QVT-R standard is not suitable for specifying even simple examples of multidirectional transformations. We intend to explore the expressive power of our multidirectional semantics for writing more realistic examples of feature model synchronization and co-evolution.

In the present paper, we have purposely left out subjective considerations about the most adequate syntactic extensions to the QVT-R language for expressing our proposed checking dependencies, and have focused primarily on the multidirectional semantics. We are currently considering several syntactic extensions to allow the specification of the checking dependencies in QVT-R.

Our `Echo` [7] model repair tool is deployed as an Eclipse plug-in that implements the bidirectional least-change technique from [6]. We plan to release a multidirectional version that naturally extends the existing one: users write multidirectional relations between models and, when inconsistencies are found, select which models are to be updated, establishing the shape of the consistency-repairing transformation.

### Acknowledgments

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by national funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FATBIT, reference FCOMP-01-0124-FEDER-020532. The first author is also sponsored by FCT grant SFRH/BD/69585/2010.

### 5. REFERENCES

- [1] J. Bradfield and P. Stevens. Enforcing QVT-R with mu-calculus and games. In *FASE'13*, volume 7793 of *LNCS*, pages 282–296. Springer, 2013.
- [2] A. Cunha, N. Macedo, and T. Guimarães. Target oriented relational model finding. In *FASE'14*, *LNCS*. Springer, 2014. To appear.
- [3] K. Czarnecki, J. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *ICMT'09*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.
- [4] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, revised edition, 2012.
- [5] F. Jouault and I. Kurtev. Transforming models with ATL. In *MoDELS'05 Satellite Events*, volume 3844 of *LNCS*, pages 128–138. Springer, 2005.
- [6] N. Macedo and A. Cunha. Implementing QVT-R bidirectional model transformations using Alloy. In *FASE'13*, volume 7793 of *LNCS*, pages 297 – 311. Springer, 2013.
- [7] N. Macedo, T. Guimarães, and A. Cunha. Model repair and transformation with Echo. In *ASE'13*, pages 694–697. IEEE, 2013.
- [8] L. Meertens. Designing constraint maintainers for user interaction. In *Third Workshop on Programmable Structured Documents*. Tokyo University, 2005.
- [9] OMG. MOF 2.0 Query/View/Transformation specification (QVT), version 1.1. <http://www.omg.org/spec/QVT/1.1/>, January 2011.
- [10] P. Stevens. A simple game-theoretic approach to checkonly QVT relations. *Software and System Modeling*, 12(1):175–199, 2013.

# Formalizing Semantic Bidirectionalization with Dependent Types

Helmut Grohne  
University of Bonn  
grohne@cs.uni-bonn.de

Andres Löh  
Well-Typed LLP  
andres@well-typed.com

Janis Voigtländer  
University of Bonn  
jv@cs.uni-bonn.de

## ABSTRACT

Bidirectionalization is the task of automatically inferring one of two transformations that as a pair realize the forward and backward relationship between two domains, subject to certain consistency conditions. A specific technique, semantic bidirectionalization, has been developed that takes a *get*-function (mapping forwards from sources to views) as input—but does not inspect its syntactic definition—and constructs a *put*-function (mapping an original source and an updated view back to an updated source), guaranteeing standard well-behavedness conditions. Proofs of the latter have been done by hand in the original paper, and recently published extensions of the technique have also come with more or less rigorous proofs or sketches thereof.

In this paper we report on a formalization of the original technique in a dependently typed programming language (turned proof assistant). This yields a complete correctness proof, with no details left out. Besides demonstrating the viability of such a completely formal approach to bidirectionalization, we see further benefits:

1. Exploration of variations of the original technique could use our formalization as a base line, providing assurance about preservation of the well-behavedness properties as one makes adjustments.
2. Thanks to being presented in a very expressive type theory, the formalization itself already provides more information about the base technique than the original work. Specifically, while the original by-hand proofs established only a partial correctness result, useful preconditions for total correctness come out of the mechanized formalization.
3. Finally, also thanks to the very precise types, there is potential for generally improving the bidirectionalization technique itself. Particularly, shape-changing updates are known to be problematic for semantic bidirectionalization, but a refined technique could leverage the information about the relationship between the shapes of sources and views now being expressed at the type level, in a way we very briefly sketch and plan to explore further.

## 1. INTRODUCTION

We are interested here in well-behaved, state-based, asymmetric lenses, in which both transformation parts of the BX are total functions. Formally, let  $S, V$  be sets. A lens in the above sense is a pair of total functions  $get: S \rightarrow V$  and  $put: S \times V \rightarrow S$  for which the following two properties hold:

$$\begin{aligned} \forall s \in S. \quad put(s, get(s)) &= s && \text{(GetPut)} \\ \forall s \in S, v \in V. \quad get(put(s, v)) &= v && \text{(PutGet)} \end{aligned}$$

Specifically, we are interested in the case when *get* is a program in a pure functional programming language and *put* is another program in the same language that is automatically obtained from *get* somehow.

Voigtländer (2009) presented a concrete technique, semantic bidirectionalization, that lets the programmer write *get* in Haskell and delivers a suitable *put* for it. The technique is both general and restricted: general in that it works independently of the syntactic definition of *get*, and restricted in that it requires *get* to have a certain (parametrically polymorphic) type. Also, it comes at the price of partiality: even when *get* is indeed a total function, the delivered *put* is in general partial; and while GetPut indeed holds as given above, PutGet becomes conditioned by *put*(*s*, *v*) actually being defined. Recent works have extended semantic bidirectionalization in various ways (Matsuda and Wang, 2013, Voigtländer et al., 2013, Wang and Najd, 2014), both to make it applicable to more *get*-functions (lifting restrictions on *get*'s type, thus allowing more varied behavior) and to make *put* (for a given *get*) defined on more inputs.

The original paper by Voigtländer (2009) gives proofs of the base technique, and papers about extensions of the technique also come with formal statements about correctness (i.e., about satisfying GetPut and PutGet) and proofs or proof sketches thereof. As is typical for by-hand proofs, details are left out and the reader is asked to believe that certain lemmas that are not explicitly proved do indeed hold and could in principle be proved by standard but tedious means. In the programming languages community there is a movement towards working more rigorously by using mechanized proof assistants to establish properties of programs (and of programming languages) in a fully formal way, see for example the POPLMARK challenge (Aydemir et al., 2005). We report here on applying this way of thinking to the semantic bidirectionalization technique, which has led to a complete formalization (Grohne, 2013) that moreover provides more precision concerning definedness of *put* than the previous proofs. The proof assistant we use is Agda, which at the same time is a pure functional programming language with

an even more expressive type system than Haskell, and we take off from there to discuss further potential such expressivity has in making semantic bidirectionalization itself more useful.

## 2. LANGUAGE

Agda is what is called a dependently typed programming language. It is a descendant of Haskell, and it is implemented in and syntactically similar to Haskell. Based, like Haskell, on a typed  $\lambda$ -calculus, Agda additionally allows values to occur as parameters to types. This mixing of types and values enables us to encode properties into types, and thus the type checker is able to verify the correctness of proofs: statements are represented by types and a proof is represented by a term that has the desired type. For this to work out, a strong discipline is required so that the type checker’s logic remains consistent; in particular, all functions must be total—runtime errors as well as non-termination of programs are ruled out by a combination of syntactic means and type checking rules. We give a brief introduction to the language; a more comprehensive account is given by Norell (2008).

As mentioned, the line between types and values is blurred in a dependently typed language. As a first example, let us have a look at the identity function. We use a slightly simplified version of the definition from the standard library<sup>1</sup>.

```
id : {α : Set} → α → α
id x = x
```

While the definition itself looks much the same as in any functional language, the type declaration is different from what one would have in Haskell, for example. That is because the availability of dependent types changes the way to express polymorphism. Instead of some convention treating certain names in a type (say, all lowercase identifiers) as type variables, we explicitly say here that  $\alpha$  shall be an element of **Set**. The type **Set** contains all types that we will use, except for itself.<sup>2</sup>

The next notable difference in the type signature of `id` is the use of curly parentheses and the fact that it has two parameters instead of one. A parameter enclosed in curly parentheses is called *implicit*. When the function is defined or used, implicit parameters are not named or given. Instead, the type system is supposed to figure out the values of these parameters. In the case of the identity function, the type of the explicit parameter will be the value of the implicit parameter. It is possible to define functions for which the type system cannot determine the values of implicit parameters. A type error will be caused in the application of such a function.

For brevity, we can declare multiple consecutive parameters of the same type without repeating the type, as can be seen in the constant function as given in the standard library<sup>3</sup>.

```
const : {α β : Set} → α → β → α
const x _ = x
```

<sup>1</sup>The `id` function is available in the `Function` module. Further footnotes about the origin of functions or types just mention the module name.

<sup>2</sup>Actually, Agda knows about a type that contains **Set**, but we are not interested in it and further types outside **Set**. Therefore, all citations from the standard library have their support for types beyond **Set** removed. Eliding those types allows us to give shorter type signatures.

<sup>3</sup>`Function`

The underscore serves as a placeholder for parameters we do not care about.

Even though the identity and constant functions already use dependent types, these examples do not illustrate the benefits of this language feature. To that end, we will have a look at functions on the data types **Fin** and **Vec** soon. Data types are introduced by notation as follows.

```
data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ
```

This definition introduces the type of natural numbers as given in the standard library<sup>4</sup>. This type is named  $\mathbb{N}$ , is an element of **Set** and takes no arguments. It has two constructors, named `zero` and `suc`, of which the latter takes a natural number as a constructor parameter. To write down elements of this type, we use constructors like functions and apply them to the required parameters. So `zero` and `suc zero` are examples for elements of  $\mathbb{N}$ .

Let us have a look at a data type with arguments. The type of finite numbers, as given in the standard library<sup>5</sup>, takes an argument of type  $\mathbb{N}$  and contains all numbers that are smaller than the argument.

```
data Fin : ℕ → Set where
  zero : {n : ℕ} → Fin (suc n)
  suc  : {n : ℕ} → Fin n → Fin (suc n)
```

We can see that declarations of the type and of constructors have the same syntax as function declarations. The names of the constructors here are shared with the  $\mathbb{N}$  type. Overloading of names is allowed for constructors, because their types can often be inferred from the context. Therefore, the constructors of **Fin** use the `suc` constructor of  $\mathbb{N}$  in their types. Also note that the type `Fin zero` has no elements.

The type of homogeneous sequences is also given in the standard library<sup>6</sup>.

```
data List (α : Set) : Set where
  [] : List α
  _::_ : α → List α → List α
```

Underscores have a special meaning when used in symbols. They denote the places where arguments shall be given in an application. For example, the list containing just the number `zero` can be written as `zeroℕ :: []`. Here we already have to disambiguate which `zero` we are referring to.

Like the **Fin** type, the **List** type takes one argument. However, this argument is given before the colon. We need to distinguish the places of arguments, because they serve different needs. An argument given after the colon is called *data index*. Any symbols bound there are not visible in constructor type signatures. The actual values given for data indices can vary among constructors, as can be seen in the definition of **Fin**. Arguments given before the colon are called *data parameters*. They are written as a space-separated sequence, and each of them must be given a name. Symbols bound as data parameters can be used both in the types of data indices and in constructor type signatures. But no discrimination is allowed on data parameters: When declaring a constructor, they must appear unchanged in the result type of the

<sup>4</sup>`Data.Nat`

<sup>5</sup>`Data.Fin`

<sup>6</sup>`Data.List`

signature. Nor are data parameters turned into (implicit) arguments of the constructors. So functions cannot branch on them when evaluating an element of a data type.

It is also possible to combine data indices and data parameters. An example for this is the type of fixed-length homogeneous sequences as given in the standard library<sup>7</sup>.

```
data Vec (α : Set) : ℕ → Set where
  [] : Vec α zero
  _::_ : {n : ℕ} → α → Vec α n → Vec α (suc n)
```

This definition has similarity to `Fin` and `List` and employs both a data parameter and a data index. Unlike in `Fin`, the base case `[]` is (only) constructible for a `zero` index instead of a `suc n` index. So for each index value there is precisely one constructor with matching type.

When defining functions on data types, we want to branch on the constructors by *pattern matching*. A simple example is the `length` function from the standard library<sup>6</sup>.

```
length : {α : Set} → List α → ℕ
length [] = zero
length (_ :: xs) = suc (length xs)
```

Unlike in Haskell, definition clauses must not overlap. For instance, the following definition will be rejected for covering the case `zero zero` twice.

```
invalid-pattern-match : ℕ → ℕ → ℕ
invalid-pattern-match zero _ = zero
invalid-pattern-match _ zero = suc zero
```

It will also be rejected for not covering the case `(suc i) (suc j)`, since all constructor combinations must be covered to meet the totality requirement.

Let us look at a truly dependently typed function now. A common task to perform on sequences is to retrieve an element from a given position. In Haskell, this can be done using the function `(!!) :: [a] -> Int -> a`. When given a negative number or a number that exceeds the length of the list, this function fails at runtime. Such behavior is prohibited in Agda, so a literal translation of this function is not possible. Ideally, the bounds check should happen at compile time. So the `Vec` type is accompanied with a corresponding retrieval function in the standard library<sup>7</sup>, as follows.

```
lookup : {α : Set} {n : ℕ} → Fin n → Vec α n → α
lookup zero (x :: xs) = x
lookup (suc i) (x :: xs) = lookup i xs
```

In the declaration, the implicit parameter `n` is used as a type parameter in the remaining function parameters. Such appearance blends the type level and value level that are clearly separated in Haskell. As a notational remark, the arrows between parameters in a type signature can be omitted if the parameters are parenthesized. The declaration above therefore lacks the arrow separating the implicit parameters.

With the totality requirement in mind, the definition of `lookup` may seem incomplete, because we omitted the case of an empty `Vec`. But a closer look reveals that that case cannot happen. The type of `[]` is `Vec α zero`, so it can only occur when `n` is `zero`. There is no constructor for `Fin zero` however. The type checker is able to do this reasoning and recognizes that our definition actually covers all type-correct

cases. Another example in a similar spirit is the definition of the `head` function from the standard library<sup>7</sup>.

```
head : {α : Set} {n : ℕ} → Vec α (suc n) → α
head (x :: _) = x
```

The input type `Vec α (suc n)` effectively expresses that only non-empty sequences can be passed—thus, no runtime error like for the corresponding Haskell function can occur.

For further familiarization, let us look at other polymorphic functions on `Lists` and/or `Vecs`. Our first example is to skip every other element of a sequence. When implemented using `Lists`, its type and implementation closely match what we would write in Haskell.

```
sieveList : {α : Set} → List α → List α
sieveList [] = []
sieveList (x :: []) = x :: []
sieveList (x :: _ :: xs) = x :: sieveList xs
```

Writing it using `Vec` requires us to give a length expression for the result type. More precisely, we need a function that relates input length to output length, in this specific case computing the upwards rounded division by 2. It happens to be available from the standard library<sup>4</sup>.

```
[_ / 2] : ℕ → ℕ
[ zero / 2 ] = zero
[ suc zero / 2 ] = suc zero
[ suc (suc n) / 2 ] = suc [ n / 2 ]
```

Equipped with this function, we can update the type of `sieve` while retaining the implementation.

```
sieveVec : {α : Set} {n : ℕ} → Vec α n
          → Vec α [ n / 2 ]
```

As another example, we consider the function that reverses a size-indexed sequence. We can base our implementation on the dependently typed left fold as does the standard library<sup>7</sup>.

```
reverseVec : {α : Set} {n : ℕ} → Vec α n → Vec α n
reverseVec {α} = foldl (Vec α) (λ rev x → x :: rev) []
```

### 3. SEMANTIC BIDIRECTIONALIZATION

The Haskell version of semantic bidirectionalization, in its most simple form, works for functions of type `[a] -> [a]`, i.e., polymorphic *get*-functions on homogeneous lists. We want to translate the Haskell implementation of “*put* from *get*” given by Voigtländer (2009) to Agda, and redevelop the proofs of the well-behavedness lens laws in parallel. So we should first look at the type of the forward function in Agda. We can think of something like `sieve` or `reverse`, so a reasonably general type expressing both the polymorphism and the possible type-level information about lengths would look as follows:

```
get : {α : Set} {n : ℕ} → Vec α n → Vec α {! !}
```

where `{! !}` is a hole that still needs to be filled by some expression. For the sake of maximal generality, we can turn the dependence of the output length on the input length into an explicit function, thus arriving at the following type:

```
get : Σ (ℕ → ℕ)
      (λ getlen → ({α : Set} {n : ℕ}
                    → Vec α n → Vec α (getlen n)))
```

<sup>7</sup>Data.Vec



The  $\Sigma$  is notation for a dependent pair as defined in the standard library<sup>8</sup>, expressing here that there is one component that is a function from  $\mathbb{N}$  to  $\mathbb{N}$  and another component whose type depends on the former function (named `getlen`). Clearly, both `sieveVec` and `reverseVec` can be thus embedded, for suitable choices of the `getlen` function. For example, the pair  $(\lfloor \_ / 2 \rfloor, \text{sieve}_{\text{Vec}})$  has the above  $\Sigma$ -type.

That indeed every polymorphic function on homogeneous lists can be thus embedded depends on free theorems, as given by Wadler (1989). One free theorem in Haskell is that for every function of type `[a] -> [a]` the length of the returned list is independent of the contents of the passed list, instead only depending on its length. Correspondingly, for list-based `get` the correct `getlen` function can be constructively obtained, and then used to define the type of the vector-based variant of `get`. The relationship here has to do with the fact that the vector type is an ornament of the list type (Dagand and McBride, 2012, 2013, Ko and Gibbons, 2013). Another way of thinking about it is colored type-theory (Bernardy and Moulin, 2013).

Now we are in a position to give the main construction from (Voigtländer, 2009). There, it is a Haskell function named `bff` (which is a short form of “bidirectionalization for free”) with the following type:<sup>9</sup>

```
bff :: (forall a. [a] -> [a])
      -> (forall a. Eq a => [a] -> [a] -> [a])
```

Apparently, a `get`-function is turned into a `put`-function, where the latter must be allowed to compare elements for equality. The most interesting bit in Agda of course is how the type plays out. It does become quite a bit more verbose, but that verbosity is useful since the additional pieces carry important information. Without further ado, here is the Agda type for `bff`:

```
bff : {getlen :  $\mathbb{N} \rightarrow \mathbb{N}$ }
       $\rightarrow (\{\alpha : \text{Set}\} \{n : \mathbb{N}\} \rightarrow \text{Vec } \alpha \ n$ 
           $\rightarrow \text{Vec } \alpha \ (\text{getlen } n))$ 
       $\rightarrow \{n : \mathbb{N}\} \rightarrow \text{Vec Carrier } n$ 
           $\rightarrow \text{Vec Carrier } (\text{getlen } n)$ 
           $\rightarrow \text{Maybe } (\text{Vec Carrier } n)$ 
```

Let us discuss this type a bit. First of all note how the dependent pair from the above prototypical Agda type for `get`, which has to take the role of the `(forall a. [a] -> [a])` argument function in Haskell’s `bff`, is turned into two arguments for `bff` by currying. For the produced `put`, instead of quantifying over an `Eq`-constrained type variable, we use a `Carrier` type that is a parameter of the Agda module in which `bff` is defined. That is solely done for convenience—since a client of the module can pass an arbitrary type for that parameter, as long as a decidable semantic equality<sup>10</sup> is defined for that type, there is no less flexibility when applying the outcome `put`-function of `bff` than there is in the Haskell

<sup>8</sup>`Data.Product`

<sup>9</sup>For simplicity, we do not yet consider type class extensions to `get`.

<sup>10</sup>To cut down on proof size, we do not support any other kind of equality at the moment. Allowing arbitrary equivalence relations here would be a first step towards supporting type class extensions to `get`. Different notions of equality/equivalence also play an important role in the work of Wang and Najd (2014) on streamlining semantic bidirectionalization for `get`-functions that are type class aware, or indeed generally higher-order.

case. Another notable difference is that the final outcome is wrapped in a `Maybe`. The reason for this is that in Agda all functions must be total. So while the Haskell implementation fails with a runtime error if no suitable result can be produced by `put`, in Agda we instead need to explicitly signal error cases as special values. Finally, the vector lengths in the type of the produced `put`-function tell us about shape constraints. In fact, mismatches between expected shape (from the original view obtained from the original source) and actual shape (from the updated view) are one reason for runtime errors in the Haskell version of `bff`. In Agda, trying to combine a source `s` that has type `Vec Carrier n` for some natural number `n` with a view `v` that has any other type than `Vec Carrier (getlen n)`, in particular one that has any other length than the expected `getlen n`, will not even be type-correct—so a possible runtime error has been turned into a static check.

The actual definition of `bff` is not much different than in Haskell. Apart from functions from the standard library<sup>11</sup> it uses a few custom functions. In particular,

```
enumerate : {n :  $\mathbb{N}$ }  $\rightarrow \text{Vec Carrier } n \rightarrow \text{Vec } (\text{Fin } n) \ n$ 
enumerate _ = tabulate id
```

enumerates the elements of a `Vec`, i.e., takes a vector of length `n` and produces a vector that corresponds to the list  $[0, 1, \dots, n-1]$ , and

```
denumerate : {n :  $\mathbb{N}$ }  $\rightarrow \text{Vec Carrier } n \rightarrow$ 
              $\text{Fin } n \rightarrow \text{Carrier}$ 
denumerate = flip lookup
```

recovers the actual values, given a position. Using some further auxiliary functions we do not repeat from (Grohne, 2013) in full here, we arrive at:

```
FinMapMaybe :  $\mathbb{N} \rightarrow \text{Set} \rightarrow \text{Set}$ 
FinMapMaybe m  $\alpha$  = Vec (Maybe  $\alpha$ ) m
checkInsert : {m :  $\mathbb{N}$ }  $\rightarrow \text{Fin } m \rightarrow \text{Carrier}$ 
               $\rightarrow \text{FinMapMaybe } m \ \text{Carrier}$ 
               $\rightarrow \text{Maybe } (\text{FinMapMaybe } m \ \text{Carrier})$ 
checkInsert i b h with lookup i h
...           | nothing = just (insert i b h)
...           | just c with deq b c
...           | yes b $\equiv$ c = just h
...           | no b $\neq$ c = nothing
assoc : {n m :  $\mathbb{N}$ }  $\rightarrow \text{Vec } (\text{Fin } m) \ n \rightarrow \text{Vec Carrier } n$ 
         $\rightarrow \text{Maybe } (\text{FinMapMaybe } m \ \text{Carrier})$ 
assoc {zero} [] [] = just empty
assoc {suc n} (i :: is) (b :: bs) = assoc is bs
                                 $\gg$  checkInsert i b
bff get s v = let s' = enumerate s
              g = tabulate (denumerate s)
              h = assoc (get s') v
              h' = (flip union g) <$> h
              in (flip mapVec s'  $\circ$  flip lookup) <$> h'
```

We do not explain all syntax used here, in particular the generalized form of pattern matching via `with`. Beside the

<sup>11</sup>For example, `flip` :  $\{\alpha \ \beta \ \gamma : \text{Set}\} \rightarrow (\alpha \rightarrow \beta \rightarrow \gamma) \rightarrow \beta \rightarrow \alpha \rightarrow \gamma$ , `mapVec` :  $\{\alpha \ \beta : \text{Set}\} \{n : \mathbb{N}\} \rightarrow (\alpha \rightarrow \beta) \rightarrow \text{Vec } \alpha \ n \rightarrow \text{Vec } \beta \ n$ , and `<$>` :  $\{\alpha \ \beta : \text{Set}\} \rightarrow (\alpha \rightarrow \beta) \rightarrow \text{Maybe } \alpha \rightarrow \text{Maybe } \beta$  are similar to their Haskell counterparts.

fact that apart from the more informative types these function definitions are rather close to those from (Voigtländer, 2009), the more interesting aspect is anyway what we can *prove* about them.

#### 4. PROVING CORRECTNESS

Voigtländer (2009) proves two theorems about `bff`, corresponding to `GetPut` and `PutGet`. In Agda, a theorem is represented/encoded as a type and a proof is a term that has that type. The two theorems as expressed in Agda are:

```
theorem-1 :
  {getlen : ℕ → ℕ}
  → (get : {α : Set} {n : ℕ} → Vec α n
      → Vec α (getlen n))
  → {n : ℕ}
  → (s : Vec Carrier n)
  → bff get s (get s) ≡ just s
```

and:

```
theorem-2 :
  {getlen : ℕ → ℕ}
  → (get : {α : Set} {n : ℕ} → Vec α n
      → Vec α (getlen n))
  → {n : ℕ}
  → (s : Vec Carrier n)
  → (v : Vec Carrier (getlen n))
  → (u : Vec Carrier n)
  → bff get s v ≡ just u
  → get u ≡ v
```

Note how both are first “quantified”—since an argument type means a piece that the user of the theorem can choose freely as long as being type-correct—over the ingredients (a `getlen` and a `get`) that are the main inputs to `bff`. Then, `theorem-1` expresses that for every `s` and every `put` obtained as `bff get` holds: `put s (get s) ≡ just s`, i.e., the here appropriate version of the `GetPut` law  $put(s, get(s)) = s$ . Similarly, `theorem-2` expresses that for every `s`, `v`, `u`, if `bff get s v ≡ just u` (note that a precondition simply becomes a function argument whose type is a statement, and thus whose every value witness will be a proof object for that statement), then `get u ≡ v`. In other words, again for `put` obtained as `bff get`: if there is some `u` such that `put s v ≡ just u`, then `get` of that `u` is `v`. That of course corresponds to the `PutGet` law,  $get(put(s, v)) = v$ , conditioned by  $put(s, v)$  actually being defined.

Complete proof objects for `theorem-1` and `theorem-2` are given in (Grohne, 2013, Agda source at <http://subdivi.de/~helmut/academia/fsbxia.agda>). We will not give those proofs/terms here; the important thing is that they exist. What is interesting to record, of course, is what assumptions they depend on. The only dependency that is *not* proved within said formalization itself is the `Vec` variant of the free theorem for polymorphic functions on homogeneous lists. Instead, it is only postulated.

```
postulate
  free-theoremVec :
    {getlen : ℕ → ℕ}
    → (get : {α : Set} {n : ℕ} → Vec α n
        → Vec α (getlen n))
    → {β γ : Set}
    → (f : β → γ) → {n : ℕ} → (l : Vec β n)
    → get (mapVec f l) ≡ mapVec f (get l)
```

This is the natural transfer of the free theorem statement for lists from Wadler (1989) to the setting of vectors. Actually proving it in Agda as well would require techniques that are orthogonal to our consideration of the lens laws (Bernardy et al., 2012), so we opt for keeping it as a postulation here, just as the list version of that free theorem for Haskell was an assumption (by all beliefs of the Haskell community a very well-founded one) in the proofs of Voigtländer (2009). The important thing is that the proofs of `theorem-1` and `theorem-2` from `free-theoremVec` are now fully machine-checked!

Those proofs themselves proceed via a series of lemmas, similarly as one would do on paper, but of course Agda is uncompromising in requiring an explicit argument for each step. There is no “this is obvious” or “left as an exercise to the reader” as in (Voigtländer, 2009) and other papers on semantic bidirectionalization and extensions thereof. Just to give a taste, here are statements that we encounter which correspond to Lemmas 1 and 2 of Voigtländer (2009):

```
lemma-1 :
  {m n : ℕ}
  → (is : Vec (Fin m) n) → (f : Fin m → Carrier)
  → assoc is (mapVec f is) ≡ just (restrict f (toList is))
```

```
lemma-2 :
  {m n : ℕ}
  → (is : Vec (Fin m) n) → (v : Vec Carrier n)
  → (h : FinMapMaybe m Carrier)
  → assoc is v ≡ just h
  → mapVec (flip lookup h) is ≡ mapVec just v
```

as well as how an induction proof in Agda looks like, for the former:<sup>12</sup>

```
lemma-1 [] f = refl
lemma-1 (i :: is) f = begin
  (assoc is (mapVec f is) ≫≡ checkInsert i (f i))
  ≡⟨ cong (λ h → h ≫≡ checkInsert i (f i))
      (lemma-1 is f) ⟩
  (just (restrict f (toList is)) ≫≡ checkInsert i (f i))
  ≡⟨ refl ⟩
  checkInsert i (f i) (restrict f (toList is))
  ≡⟨ lemma-checkInsert-restrict f i (toList is) ⟩
  just (insert i (f i) (restrict f (toList is))) □
```

farming out to another auxiliary lemma:

```
lemma-checkInsert-restrict :
  {m : ℕ}
  → (f : Fin m → Carrier)
  → (i : Fin m) → (is : List (Fin m))
  → checkInsert i (f i) (restrict f is)
  ≡ just (restrict f (i :: is))
```

which in turn requires further inductions, etc. Something we do not dwell on here is the actual *process* of arriving at the proofs, but Grohne (2013) describes in detail how interactive proof construction works and how Agda lends a helping hand, while also requiring familiarization with certain idioms for effective formalization. This guidance

<sup>12</sup>The `refl` steps correspond to reflexivity of propositional equality  $\equiv$ . It can be used when Agda is able to prove an equality by its built-in rewriting strategy based on function definitions. Such rewriting also happens silently, but of course always with Agda’s correctness guarantee, in some other steps.

should be helpful when embarking on a similar endeavor for correctness proofs of other techniques, or when further developing the provided formalization, to cover extensions of semantic bidirectionalization already presented in the literature or still to be explored.

## 5. SO WHAT?

We have arrived at formal proofs of GetPut and PutGet for the bidirectionalization technique from (Voigtländer, 2009). But we already knew, or at least very strongly believed, that the technique was correct beforehand. After all, the original paper did contain lemmas, theorems, and proofs that seemed acceptable to the community. So what have we actually gained?

Beside the reassuring feeling that comes with a machine-checked proof, the dependent types and formalization work bring concrete additional benefits in terms of better understanding of the formalized technique and its properties. We have already remarked on the fact that the Haskell version of `bff` can fail with a runtime error, and that one reason for such failure is shape mismatches, and that the constraints on vector lengths in the Agda types we use prevent those. Actually, it was already informally observed in previous work for the Haskell version that only when the shapes of `get s` and `v` are the same is there any hope that `put s v` is defined, but the dependent types in the Agda version are both explicit and more rigorous about this.

And there is more. Even when the shapes are in the correct relationship, the `put` obtained as `bff get` can fail. After all, that is why we have wrapped the ultimate return type of `bff` in a `Maybe`. Such failure occurs when `get` duplicates some entry from the source sequence and the two copies in the view are updated to different values. On the other hand, if no duplication takes place, then `bff` should not end up returning `nothing` (thus signaling failure). In Agda, we can formalize this intuition based on the following predicate:

```

data All-different {α : Set} : List α → Set where
  different-[] : All-different []
  different-:: : {x : α} {xs : List α}
    → x ∉ xs
    → All-different xs
    → All-different (x :: xs)

```

What this definition says is that, trivially, the elements of the empty list are pairwise different, and the elements of a non-empty list are pairwise different if the head element is not contained in the tail and if, moreover, the elements of the tail are pairwise different. Based on `All-different`, Grohne (2013) proves a sufficient condition for when an `assoc`-call succeeds (i.e., for when there exists some `h` such that the result of `assoc` is `just h` rather than `nothing`):

```

different-assoc :
  {m n : ℕ}
  → (u : Vec (Fin m) n)
  → (v : Vec Carrier n)
  → All-different (toList u)
  → ∃ (λ h → assoc u v ≡ just h)

```

Moreover, he proves that if a certain `assoc`-call succeeds, then the `put` obtained as `bff get` succeeds:

```

lemma-assoc-enough :
  {getlen : ℕ → ℕ}

```

```

→ (get : {α : Set} {n : ℕ} → Vec α n
   → Vec α (getlen n))
→ {n : ℕ}
→ (s : Vec Carrier n)
→ (v : Vec Carrier (getlen n))
→ ∃ (λ h → assoc (get (enumerate s)) v ≡ just h)
→ ∃ (λ u → bff get s v ≡ just u)

```

Combining `different-assoc` and `lemma-assoc-enough`, we learn that `bff get s v` succeeds, and thus the precondition of `PutGet/theorem-2` is fulfilled, if

All-different (toList (get (enumerate s)))

holds. Thus, we have formally established that a sufficient condition on `get` to guarantee that the dependently typed `bff get` always succeeds is what is called *semantically affine* in (Voigtländer et al., 2013).

Further exploration of semantic bidirectionalization techniques should also profit from the availability of a formalization. Indeed, such availability would have benefited us in the past. For example, the original paper (Voigtländer, 2009) proved `GetPut` and `PutGet`, but only claimed that a third law, `PutPut`, also holds. Later work (Foster et al., 2012) refactored the definition of `bff`, essentially by formulating it in terms of the constant-complement approach (Bancilhon and Spyrtos, 1981), to make more apparent that `PutPut` indeed holds. But this refactoring required extra care and consideration to make sure that no other properties were destroyed. In fact, new arguments were needed for correctness of the refactored version. Of course, the same would have been the case if an Agda formalization of the original correctness arguments had already been available, but the dependent types and proof assistant would have provided a safety net, just as standard type systems provide a safety net when refactoring ordinary programs instead of programs and proofs in one go. Similarly, other and further variations of semantic bidirectionalization may profit now. It would be useful to first extend the formalization to treat data structures other than sequences for `get` to operate on, for example trees. Data type generic versions of `get` have already been implemented in Haskell Voigtländer (2009), Foster et al. (2012), but not been proved with the same rigor. The formalization of indexed containers using ornaments (Dagand and McBride, 2013) should be useful here.

Finally, let us mention a promising new direction for bidirectionalization that uses dependent types not only for verification but for doing a better job at the bidirectionalization task itself. The idea here is to turn dependent types into a “plug-in” in the sense of (Voigtländer et al., 2013). In brief, the variation of semantic bidirectionalization presented by Voigtländer et al. (2013) overcomes the limitation of only being able to handle shape-preserving updates. It does so by requiring that each invocation of `bff` is enriched by a “shape bidirectionalizer”, a function that performs well-behaved updates on an abstraction of sources and views to the shape level, for example list lengths. Several possibilities are discussed for solving the shape-level problem, ranging from requesting programmer input, over search and syntactic transformations, to bootstrapping semantic bidirectionalization for abstracted problems. All this happens in Haskell, but in Agda we have another resource for such plug-in techniques. Namely, we can turn to shape information that comes from the types. Specifically, the `getlen` functions already express relationships between source and view sequence lengths.

Since the propagation direction needed for shape bidirectionalizer plug-ins is from views to sources, we would actually need at least a partial inverse of `getlen`. But with the rich expressiveness available at the type level in Agda, we could even explore different abstractions, be they general relations between source and view shapes, or functions in one or the other direction. We can also prove connections between these abstractions, and potentially move between them, depending on what is most convenient for a given `get`-function. As a very simple example of what we have in mind, consider the `tail` function with its canonical type in Agda:

$$\begin{aligned} \text{tail} &: \{ \alpha : \text{Set} \} \{ n : \mathbb{N} \} \rightarrow \text{Vec } \alpha \text{ (suc } n) \rightarrow \text{Vec } \alpha \ n \\ \text{tail } (- :: xs) &= xs \end{aligned}$$

The type does not only express that `tail` is only well-defined on non-empty sequences, it also tells us in no uncertain terms that its input is always exactly one entry longer than its output (so `suc` acts as `getlen`<sup>-1</sup> here). Concerning bidirectionality that tells us that if `tail` is `get` and the view sequence is changed to some new length, we know exactly what the new source length should be. This is exactly the information that a shape bidirectionalizer plug-in needs to provide, but now actually available statically by virtue of the very definition of `get` in a dependently typed language. We plan to develop a general technique from this idea, of course with Agda implementation and formalization going hand in hand.

## References

- Brian E. Aydemir, Aaron Bohannon, Matthew Fairbairn, J. Nathan Foster, Benjamin C. Pierce, Peter Sewell, Dimitrios Vytiniotis, Geoffrey Washburn, Stephanie Weirich, and Steve Zdancewic. Mechanized metatheory for the masses: The POPLMARK challenge. In *Proceedings of Theorem Proving in Higher Order Logics*, volume 3603 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2005. doi 10.1007/11541868\_4.
- François Bancilhon and Nicolas Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, 1981. doi 10.1145/319628.319634.
- Jean-Philippe Bernardy and Guilhem Moulin. Type-theory in color. In *Proceedings of International Conference on Functional Programming*, pages 61–72. ACM, 2013. doi 10.1145/2500365.2500577.
- Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free—Parametricity for dependent types. *Journal of Functional Programming*, 22(2):107–152, 2012. doi 10.1017/S0956796812000056.
- Pierre-Évariste Dagand and Conor McBride. Transporting functions across ornaments. In *Proceedings of International Conference on Functional Programming*, pages 103–114. ACM, 2012. doi 10.1145/2364527.2364544.
- Pierre-Évariste Dagand and Conor McBride. A categorical treatment of ornaments. In *Proceedings of Logic in Computer Science*, pages 530–539. IEEE, 2013. doi 10.1109/LICS.2013.60.
- Nils Anders Danielsson et al. The Agda standard library version 0.6, 2011. URL <http://www.cse.chalmers.se/~nad/software/lib-0.6.tar.gz>.
- Nate Foster, Kazutaka Matsuda, and Janis Voigtländer. Three complementary approaches to bidirectional programming. In *Spring School on Generic and Indexed Programming (SSGIP 2010), Revised Lectures*, volume 7470 of *Lecture Notes in Computer Science*, pages 1–46. Springer, 2012. doi 10.1007/978-3-642-32202-0\_1.
- Helmut Grohne. Formalizing semantic bidirectionalization in Agda. Master’s thesis, University of Bonn, 2013. URL <http://subdivi.de/~helmut/academia/fsbxia.pdf>. Agda formalization available at <http://subdivi.de/~helmut/academia/fsbxia.agda>.
- Hsiang-Shang Ko and Jeremy Gibbons. Modularising inductive families. *Progress in Informatics*, 10:65–88, 2013. doi 10.2201/NiiPi.2013.10.5.
- Kazutaka Matsuda and Meng Wang. Bidirectionalization for free with runtime recording: Or, a light-weight approach to the view-update problem. In *Proceedings of Principles and Practice of Declarative Programming*, pages 297–308. ACM, 2013. doi 10.1145/2505879.2505888.
- Ulf Norell. Dependently typed programming in Agda. In *Advanced Functional Programming*, volume 5832 of *Lecture Notes in Computer Science*, pages 230–266. Springer, 2008. doi 10.1007/978-3-642-04652-0\_5.
- Janis Voigtländer. Bidirectionalization for free! (Pearl). In *Proceedings of Principles of Programming Languages*, pages 165–176. ACM, 2009. doi 10.1145/1480881.1480904.
- Janis Voigtländer, Zhenjiang Hu, Kazutaka Matsuda, and Meng Wang. Enhancing semantic bidirectionalization via shape bidirectionalizer plug-ins. *Journal of Functional Programming*, 23(5):515–551, 2013. doi 10.1017/S0956796813000130.
- Philip Wadler. Theorems for free! In *Proceedings of Functional Programming languages and Computer Architecture*, pages 347–359. ACM, 1989. doi 10.1145/99370.99404.
- Meng Wang and Shayan Najd. Semantic bidirectionalization revisited. In *Proceedings of Partial Evaluation and Program Manipulation*, pages 51–61. ACM, 2014. doi 10.1145/2543728.2543729.

# BenchmarkX

Anthony Anjorin  
Technische Universität  
Darmstadt  
anthony.anjorin@es.tu-  
darmstadt.de

Frank Hermann  
Université du Luxembourg  
Interdisciplinary Centre for  
Security, Reliability and Trust  
frank.hermann@uni.lu

Alcino Cunha  
HASLab / INESC TEC and  
Universidade do Minho  
alcino@di.uminho.pt

Arend Rensink  
University of Twente  
arend.rensink@utwente.nl

Holger Giese  
Hasso-Plattner-Institut  
holger.giese@  
hp.uni-potsdam.de

Andy Schürr  
Technische Universität  
Darmstadt  
andy.schuerr@es.tu-  
darmstadt.de

## ABSTRACT

Bidirectional transformation (BX) is a very active area of research interest. There is not only a growing body of theory, but also a rich set of tools supporting BX. The problem now arises that there is no commonly agreed-upon suite of tests or benchmarks that shows either the conformance of tools to theory, or the performance of tools in particular BX scenarios. This paper sets out to improve the state of affairs in this respect, by proposing a template and a set of required criteria for benchmark descriptions, as well as guidelines for the artifacts that should be provided for each included test. As a proof of concept, the paper additionally provides a detailed description of one concrete benchmark.

## Categories and Subject Descriptors

D.4.8 [Software Engineering]: Performance—Measurements

## General Terms

Measurement

## Keywords

bidirectional transformations, benchmark, tools, comparison

## 1. INTRODUCTION AND MOTIVATION

Bidirectional transformations (BX) are required to maintain the consistency of related artifacts modified by concurrent engineering activities [2]. This task is relevant in multiple domains and is an active research focus in various communities [2]. Although the first theoretical foundations for BX have been laid and there are a number of tools that already support BX to a certain extent [9], it is difficult for non-developers to discern the exact capabilities of each BX tool and effectively compare it with others.

What is missing is a collection of *benchmarks*, which can be used to identify the strengths and weaknesses of different tools and their

respective approaches. Such a benchmark suite for BX will not only clarify the state of the art and current limitations of BX tools, but also drive further development and cross-fertilization between the different BX sub-communities and tool developers.

Our goal in this paper is to improve the current situation by proposing a template for BX benchmarks. Specifically, we:

1. Provide a precise definition for a *BX benchmark*, identifying a set of required properties that distinguish a BX benchmark from a mere BX example.
2. Identify a *feature matrix* similar to that given in [12], which is to be used to classify BX benchmarks.
3. Present, as a *proof of concept*, a simple but concrete benchmark that adheres to our proposed template.

## 2. STRUCTURE OF A BX BENCHMARK

According to the dictionary,<sup>1</sup> a *benchmark* is a *standardised problem or test that serves as a basis for evaluation or comparison*. The aim of *benchmarking* is to systematically assess and, if possible, measure a set of features of a system under different, precisely defined, and reproducible circumstances [12]. Based on the respective results on a benchmark, different systems can be compared and evaluated based on pertinent system characteristics. In this section we collect the characteristics needed for a BX benchmark to make such assessment and measurement possible.

### 2.1 Prerequisites

In order to precisely define different aspects and properties of bidirectional transformations in this paper, we first provide some detailed formalisations and general requirements.

A *bidirectional transformation* consists of the following:

- A *left language*  $\mathcal{L}^L$ , a *right language*  $\mathcal{L}^R$ , and a binary *consistency relation*  $C$  over both languages that specifies which pairs of left and right models are consistent. It is difficult to escape the directional cultural bias here, though we try to avoid equating “left” with “source” and “right” with “target”.
- Sets of possible left updates  $U^L$  and right updates  $U^R$ . Each update  $u$  is a mapping from models to corresponding updated models. The application of an update to a single model is given by a *change*  $\delta$  (we shall state exactly what this is in a moment), that specifies how a model is modified to a changed model. The sets of possible changes to the left and right models are denoted by  $\Delta^L$  and  $\Delta^R$ , respectively.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

<sup>1</sup>Merriam-Webster, 2013

- Propagation functions  $\vec{C}: \Delta^L \times \mathcal{L}^R \rightarrow \Delta^R$  and  $\overleftarrow{C}: \Delta^R \times \mathcal{L}^L \rightarrow \Delta^L$ , commonly called *forward* and *backward* propagation. The forward propagation  $\vec{C}$  takes a change  $\delta$  on the left-side together with a right model (consistent with the left model that is the source of  $\delta$ ), and returns a change on the right-side to be applied to that model. The same applies dually for the backward propagation  $\overleftarrow{C}$ .

We will equate languages with their extensions, being sets of *models*; hence we can regard  $C \subseteq \mathcal{L}^L \times \mathcal{L}^R$  as a binary relation between left models and right models. Consistency of a left-model  $M^L \in \mathcal{L}^L$  and a right-model  $M^R \in \mathcal{L}^R$  will be denoted by  $M^L C M^R$ . In some scenarios, the consistency relation  $C$  may only be admitted if there is further evidence of consistency, for instance in the form of an element-to-element mapping between related models; this can be captured by defining  $C$  as the union of a set of *indexed* binary relations  $C_\varphi$ , where  $\varphi$  is the object that captures the evidence — for instance, a binary relation between the elements of the left model and those of the right model.

Members of the sets of left updates  $U^L$  and right updates  $U^R$  are partial functions from models to models. For instance, a left update  $u^L \in U^L$  is a partial function from  $\mathcal{L}^L$  to  $\mathcal{L}^L$ . We reserve the word *change* (or *delta*) for the pairs of models that are the extension of an update; i.e., if  $u(M_1) = M_2$  for some update  $u$  then  $(M_1, M_2)$  is a change. Like pairs of consistent models, in some scenarios a change may have additional information about the relation between the elements of source and target model; in such a situation, the change will be a triple  $(M_1, M_2, \mu)$  for some object  $\mu$  that encodes the additional information. In general, a change  $\delta$  will always have a well-defined source model  $\text{src}(\delta)$  and target model  $\text{tgt}(\delta)$ . The set of all allowed left- [right-] changes is denoted by  $\Delta^L$  [ $\Delta^R$ ]. A single change  $\delta$  may be taken as a kind of degenerate update  $u$ , applicable only to  $\text{src}(\delta)$  and yielding  $\text{tgt}(\delta)$ .

The general requirement for  $\vec{C}$  is that for all  $\delta^L \in \Delta^L$  and  $M^R \in \mathcal{L}^R$ , such that  $\text{src}(\delta^L) C M^R$ :

$$M^R = \text{src}(\vec{C}(\delta^L, M^R)) \quad (1)$$

and analogously for  $\overleftarrow{C}$ . These requirements are usually known as *incidence laws* in delta-based BX frameworks [3]. In the simplest case, where  $\Delta = \mathcal{L} \times \mathcal{L}$  is the set of all possible pairs of models, (1) can be trivially satisfied. It should be noted that, when  $C$  is left-unique and left-total (see below), then the target of  $\vec{C}(\delta^L, M^R)$  is uniquely defined by the target of  $\delta^L$ ; and dually for the other direction. A special case of (say forward) propagation is where  $\delta^L$  is the change from some initial, empty model  $\perp \in \mathcal{L}^L$  into our left model of interest  $M^L$  (often referred to as a *batch* transformation).

How forward and backward propagation are defined is up to the benchmark in question. For instance, if there is additional *evidence*  $C_\varphi$  associated with the consistency of  $M^L$  and  $M^R$ , then  $C_\varphi$  might play a role in the definition of  $\vec{C}$  and  $\overleftarrow{C}$  (and be required as input).

## 2.2 Definition of a BX benchmark

The following definitions are adapted from [12] and adjusted as required for BX benchmarks:

A *BX scenario* is a broad application field that can be clearly characterized as requiring BX. Examples include model synchronization, tool integration, and round-trip engineering.

A *BX benchmark* is a bidirectional transformation that serves as an incarnation of a BX scenario, with the following properties:

1. A precise, *executable* definition of the consistency relation  $C$ , which can be used as an oracle to decide if a left model  $M^L \in \mathcal{L}^L$ , and a right model  $M^R \in \mathcal{L}^R$  are consistent (i.e.,  $M^L C M^R$ ).

2. An explicit definition of data (input models) for the transformation, or a *generator* that can be used to produce the required models.
3. A set of precisely defined updates for certain input models.
4. A set of *executable* metric definitions for what is to be measured / assessed by the benchmark.
5. Finally, as it is of no interest to measure arbitrary and irrelevant features, a benchmark should represent a *useful* transformation that covers aspects that are indeed relevant in the corresponding BX scenario.

A BX benchmark consists of several *test cases*, each of which is a complete, deterministic, but parametric specification fixing all details required for executing the corresponding measurements. Every test case measures or assesses specific features.

Finally, a *run* is a test case for which all parameters (e.g., input data size) are set to concrete values.

The results of carrying out a benchmark for a specific tool are produced by executing a series of runs for the different test cases of the benchmark.

## 2.3 Classification of BX benchmarks

A BX benchmark is to be classified using a *feature matrix*, with one dimension corresponding to the different test cases of the benchmark, and the other to paradigm and tool features, discussed in the following.

### 2.3.1 Paradigm features of the benchmark

A *paradigm feature* of a BX benchmark describes a characteristic of a test case of the benchmark. We identify the following paradigm features (to be extended as required):

**Properties of the consistency relation.** The following properties purely depend on the left and right languages. While in some BX approaches, the consistency relation is derived from the specification of the BX operations, we explicitly avoid this coupling. This implies that the benchmark can be seen as a test for the correctness and completeness of the particular solution using a specific approach.

1.  $C$  can be *left-total*, meaning that for every left model  $M^L \in \mathcal{L}^L$ , there exists *at least one* right model  $M^R \in \mathcal{L}^R$  such that  $M^L C M^R$ ; and *right-total*, which is the dual and defined analogously.
2.  $C$  can be *left-unique*, meaning that for every right model  $M^R \in \mathcal{L}^R$ , there exists *at most one* left model  $M^L \in \mathcal{L}^L$  such that  $M^L C M^R$ ; and *right-unique*, which is the dual and defined analogously.

For those more familiar with other terminology: a left-total relation is often called *total*, right-total *surjective*, left-unique *injective*, and right-unique *functional*.

**Platform dependency.** A test case can range from *platform independent* (PI) to *platform specific* (PS), depending on the nature of its description. Note that according to our definition of a BX benchmark, data *must* be provided, but this can be, e.g., in form of a textual format for which adapters for different platforms can be provided as required (PI), as opposed to, say, Ecore XMI files (PS). A benchmark should state explicitly the platforms for which data or appropriate adapters are provided.

**Characteristics of the data domain.** Every benchmark (or individual test case) should state the characteristics of the required data domain explicitly. With this we mean (i) if the data can be represented as simple sets, trees, or graphs, and (ii) what constraints are imposed including keys, an order on elements, etc. This is important as some BX tools are specialized for, e.g., sets and do not support graph-like structures.

**Variations and extension points.** Every benchmark should state if it is a variation of an existing benchmark and make the differences as explicit as possible. The goal here is to be able to document and manage a family of different benchmarks, which are clearly related, i.e., use basically the same transformation, but either simplify or introduce additional complexity.

### 2.3.2 Tool features measured by the benchmark

A *tool feature* of a BX benchmark represents a tool characteristic that can be assessed and measured with a series of runs of a test case of the benchmark. We identify the following tool features (to be extended as required):

**Run time:** The run time required to execute a run can be measured in, e.g., milliseconds for a tool. Note that run time can also be measured abstractly in “actions” or “edits” depending on the used technology (e.g., a database or model repository).

**Memory Consumption:** The required memory for executing a run can be measured for a tool either in (kilo)bytes or, as for run time, in more abstract units if this makes sense.

**Scalability:** By plotting run time/memory consumption for (i) increasing input data size and (ii) increasing size of propagated changes ( $U^L, U^R$ ), the scalability of a tool can be measured with respect to the varied dimension. Note that this is only feasible if the test case consists of several runs of increasing size or if a size parametric data generator is provided.

**Conformance to laws for BX approaches:** The expected behaviour of BX tools is typically assessed by checking conformance to a set of laws. Depending on the BX framework, e.g., lenses [3], Triple Graph Grammars [8], or other BX algebraic frameworks [10], various sets of laws can be formulated differently.

Given  $M^L \in \mathcal{L}^L$ ,  $M^R \in \mathcal{L}^R$ ,  $\delta^L \in \Delta^L$  such that  $M^L \mathcal{C} M^R$  and  $\text{src}(\delta^L) = M^L$ , we identify the following potential laws of interest (to be extended as required):

**Correctness.** A basic law present in all frameworks is conformance of the forward and backward propagation to the consistency relation  $\mathcal{C}$ . *Forward correctness* is stated as:

$$\left[ \vec{\mathcal{C}}(\delta^L, M^R) = \delta^R \right] \implies \text{tgt}(\delta^L) \mathcal{C} \text{tgt}(\delta^R)$$

*Backward correctness* is defined analogously.

**Completeness.** Another interesting property to assess is whether the forward propagation succeeds in returning a change whenever at least one consistent right-model exists. *Forward completeness* can be stated as:

$$\left[ \exists M_2^R . \text{tgt}(\delta^L) \mathcal{C} M_2^R \right] \implies \vec{\mathcal{C}}(\delta^L, M^R) \downarrow$$

Here,  $\vec{\mathcal{C}}(\delta^L, M^R) \downarrow$  means that the forward propagation is defined for the given inputs, i.e., the execution is successful and returns a change on the right domain. *Backward completeness* is defined analogously.

As an alternative to the more consensual notion of correctness proposed above, some scenarios may require some notion of compatibility of forward and backward propagation operations, e.g., the GETPUT and PUTGET laws [6], or (weak) invertibility [4]. For instance, instead of requiring the result of a forward propagation to be consistent, one may only require it to be constant under an additional round trip application of backward and forward propagations. Benchmarks may vary on the used laws to assess expected behaviour.

If the consistency relation  $\mathcal{C}$  is *not* source unique, backward propagation must possibly deal with a *loss of information* as it is unclear and in general impossible to reconstruct the “expected” source only given a modified target. It is important to assess if a

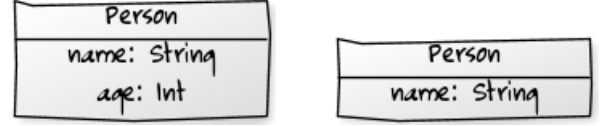


Figure 1: Meta-models for  $\mathcal{L}^L$  (left) and  $\mathcal{L}^R$  (right)

tool is able to use the old source to cope adequately with information loss. The situation is analogous for non target uniqueness and forward propagation. The following laws can be used to assess this:

**Hippocraticness.** A basic property that is present in most frameworks forbids *unnecessary* changes when the left and right model are already consistent. This is most commonly known in a simplified form as *hippocraticness* [10]. *Forward hippocraticness* can be formalized as follows (similarly for *backward hippocraticness*):

$$\left[ \text{tgt}(\delta^L) \mathcal{C} M^R \wedge \vec{\mathcal{C}}(\delta^L, M^R) = \delta^R \right] \implies \text{tgt}(\delta^R) = M^R$$

**Least Change.** A stronger requirement than hippocraticness is the *least change* property, that forbids changes that are clearly unnecessary (first introduced in [7] as the *principle of least change*). A formalization of *forward least change* is:

$$\left[ \vec{\mathcal{C}}(\delta^L, M^R) = \delta^R \right] \implies \nexists \delta_2^R . \text{src}(\delta_2^R) = M^R \wedge \text{tgt}(\delta^L) \mathcal{C} \text{tgt}(\delta_2^R) \wedge \delta_2^R <_{\Delta} \delta^R$$

Here we assume the existence of a partial order  $<_{\Delta}$  between changes that somehow measures if one change is “smaller” than another. The concrete definition of this partial order is benchmark dependent, and it may be interesting to even have different instantiations of this law for different partial orders.

## 3. AN EXAMPLE BX BENCHMARK

We presuppose primitive types **String** and **Int**. Our left and right languages are defined by the meta-models in Fig. 1. Both are extremely simple: the left language consists of *sets* of **Persons**, with name and age attributes, whereas the right language likewise consists of collections of **Persons**, in this case only with name attributes. For both languages, name is considered to be a *key* attribute, meaning that only collections of persons with distinct names are allowed. The languages are extensionally defined as sets of models with the following characteristics:

- $\mathcal{L}^L$  is the set of finite partial functions  $M^L : \mathbf{Int} \rightarrow (\mathbf{String} \times \mathbf{Int})$  such that for all pairs of elements  $M^L(p_i) = (n_i, a_i)$  for  $i = 1, 2$ ,  $n_1 = n_2$  implies  $p_1 = p_2$ .
- $\mathcal{L}^R$  is the set of all finite injective partial functions  $M^R : \mathbf{Int} \rightarrow \mathbf{String}$ .

Note that models are not simply tuples of names and ages, respectively names: they include an *identity*, here taken from the set **Int**. This is essential for the benchmark, as it allows us to speak meaningfully of a given person changing his name while remaining the same entity. The actual choice of identity does not matter. Also note that, though for conciseness we have chosen to formulate models as finite partial functions from the identity set to the corresponding tuple of attributes, this is mathematically equivalent to a left-unique relation between the identity set and tuples of attributes.

**Consistency relation.** We will use  $\text{name}^L : \mathbf{Int} \rightarrow \mathbf{String}$  [resp.  $\text{age}^L : \mathbf{Int} \rightarrow \mathbf{Int}$ ] to denote the partial function mapping each person  $p \in \text{dom}(M^L)$  to the first [second] component of  $M^L(p)$ ; likewise,  $\text{name}^R : \mathbf{Int} \rightarrow \mathbf{String}$  will have the same meaning in the right domain (hence it is actually the same function as  $M^R$ ).

The consistency relation is defined by  $C \subseteq \mathcal{L}^L \times \mathcal{L}^R$  such that

$$M^L C M^R \Leftrightarrow \exists \varphi : \text{dom}(M^L) \leftrightarrow \text{dom}(M^R), \text{name}^L = \text{name}^R \circ \varphi$$

In words: a pair of left/right models is related if and only if there is a name-preserving bijection between the person identities in the left and those in the right models. Thus,  $\varphi$  is an example of the *evidence* mentioned in the previous section.

This consistency relation exhibits the following properties:

- It is both left-total and right-total;
- It is not left-unique but it is right-unique;
- It implies a unique one-to-one relation on the element level.

**Changes.** In general, a left-change is any triple  $(M_1^L, M_2^L, \mu)$  where  $M_1^L$  and  $M_2^L$  are left models, and  $\mu : \text{dom}(M_1^L) \rightarrow \text{dom}(M_2^L)$  is a partial injective function; A right-change is the same but for right models.

The mapping  $\mu$  connects the identities used in the original model to those in the destination model. It is a partial function, meaning that identities can neither be split nor merged by a change, but a person  $p$  can be deleted (in which case  $p \in \text{dom}(M_1^L) \setminus \text{dom}(\mu)$ ) or created (in which case  $p \in \text{dom}(M_2^L) \setminus \text{cod}(\mu)$ ). In practice, identities will typically not change at all during updates, except for deletion and creation, and so  $\mu$  will always be a partial identity. Even so, it is important to recognise this component; for instance, when identities are reused (a person is deleted from the database and later another person is added, reusing the identity of the first) this does *not* mean it is the same person — and the only way in which this confusion is avoided is through  $\mu$ . In fact, in a very real sense, the presence of the mapping  $\mu$  defines the difference between state-based and delta-based BX approaches.

A left-change is *minimal* if one of the following cases holds:

1. It deletes a single element:
  - $|\text{dom}(M_1^L) \setminus \text{dom}(\mu)| = 1$  and  $\text{cod}(\mu) = \text{dom}(M_2^L)$
  - $M_2^L(\mu(p)) = M_1^L(p)$  for all  $p \in \text{dom}(\mu)$ .
2. It creates a single element:
  - $\text{dom}(M_1^L) = \text{dom}(\mu)$  and  $|\text{dom}(M_2^L) \setminus \text{cod}(\mu)| = 1$
  - $M_2^L(\mu(p)) = M_1^L(p)$  for all  $p \in \text{dom}(\mu)$ .
  - $M_2^L(p_i) = (n_i, a_i)$  for  $i = 1, 2$ ,  $n_1 = n_2$  implies  $p_1 = p_2$ .
3. It modifies one of the fields of a single element:
  - $\text{dom}(\mu) = \text{dom}(M_1^L)$  and  $\text{cod}(\mu) = \text{dom}(M_2^L)$
  - For a single  $p \in \text{dom}(\mu)$ , either
    - (a)  $\text{name}_1^L(p) \neq \text{name}_2^L(\mu(p))$  or
    - (b)  $\text{age}_1^L(p) \neq \text{age}_2^L(\mu(p))$  (but not both)
  - $M_1^L(q) = M_2^L(\mu(q))$  for all  $q \in \text{dom}(\mu) \setminus \{p\}$ .

It can be seen immediately that every change is a composition of a finite sequence of minimal changes. For each of these minimal changes we can formulate update functions that cause them:

1. For all  $s \in \mathbf{String}$ ,  $\text{delete}(s) \in U^L$  is an update function that deletes the person  $p$  with  $\text{name}^L(p) = s$ .
2. For all  $s \in \mathbf{String}$ ,  $\text{create}(s) \in U^L$  is an update function that adds a person  $p$  with  $\text{name}^L(p) = s$ .
3. (a) For all  $s_1, s_2 \in \mathbf{String}$ ,  $\text{setName}(s_1, s_2) \in U^L$  is an update function that changes the name of a person from  $s_1$  (in the source model) into  $s_2$  (in the target model).
  - (b) For all  $s \in \mathbf{String}$ ,  $a \in \mathbf{Int}$ ,  $\text{setAge}(s, a) \in U^L$  is an update function that changes the age of a person with name  $s$  to  $a$ .

The right changes and updates are defined analogously, except that, obviously, the age attribute (case 3b) is irrelevant.

**Propagation.** According to our definition of a BX benchmark, we now have to precisely define update propagation for the example. Since we have shown how arbitrary changes can be decomposed into minimal changes, we only have to explain how minimal

changes are propagated.<sup>2</sup> Forward propagation is actually already completely defined by demanding correctness as  $C$  is left-unique. We therefore concentrate on backward propagation. Let  $M_1^L C_\varphi M_1^R$  be a pair of consistent models, and let  $\delta^R = (M_1^R, M_2^R, \mu^R)$  be a minimal right-change. We define  $\delta^L = \overleftarrow{C}(\delta^R, M_1^L)$  for each of the cases of minimal change listed above. Note that age changes do not exist in this setting.

1. For deletion, let  $p \in \text{dom}(M_1^R) \setminus \text{dom}(\mu^R)$ , and define  $M_2^L$  as the restriction of  $M_1^L$  to all persons except  $\varphi^{-1}(p)$ .  $\mu^L$  is the identity function on  $\text{dom}(M_2^L)$ .
2. For creation, let  $p \in \text{dom}(M_2^R) \setminus \text{cod}(\mu^R)$  and fresh  $q \notin \text{dom}(M_1^L)$ ; define  $M_2^L = M_1^L \cup \{(q, (\text{name}_2^R(\mu^R(p)), a))\}$  for some default age  $a$ .  $\mu^L$  is the identity function on  $\text{dom}(M_1^L)$ .
3. Let  $p \in \text{dom}(M_1^R)$  be the person whose name has changed; let  $s_1 = \text{name}_1^R(p)$ ,  $a_1 = \text{age}_1^R(p)$ , and  $s_2 = \text{name}_2^R(\mu^R(p))$ , and let  $q = \varphi^{-1}(p)$ . Define  $\{M_2^L = M_1^L \setminus \{(q, (s_1, a_1))\}\} \cup \{(q, (s_2, a_1))\}$ .  $\mu^L$  is the identity function on  $\text{dom}(M_1^L)$ .

The interesting cases, for which not all information is available in the right model, are when a person is created (in that case a *default* age has to be inserted) or when a name is changed (in that case the age must be preserved).

**Test cases and BenchmarkX repository.** The draft and artifacts related to this first benchmark (to be known as `Person2Person`), and all future BenchmarkXs, will be uploaded to the BX Examples Repository, being set up by Cheney et al. [1] at the BX community Wiki at <http://bx-community.wikidot.com/examples:home>

Several illustrative test cases are already described in the Wiki, tailored for assessing different operational modes of existing BX tools, including the following:

**Person2Person.BCF** The goal of this test case is to assess the conformance to BX properties (correctness and completeness) of a forward propagation run in batch mode (i.e. to create a new right-model from an input left-model). This is equivalent to providing as original right-model the empty model with no persons. The inputs for the runs in this test case will be typically small in size, and hand-picked to expose possible corner cases where achieving conformance might not be trivial (in general this will be the case with conformance test cases).

**Person2Person.BCB** This test case is similar to the previous, but for the backward propagation.

**Person2Person.BSF** The goal of this test case is to assess the performance (scalability of run time and memory consumption) of a forward propagation run in batch mode. A size parametric left-model generator will be provided.

**Person2Person.BSB** This test case is similar to the previous, but for the backward propagation.

**Person2Person.MCB** The goal of this test case is to assess the conformance to BX properties (correctness, completeness, and hippocraticness) of state based backward propagation. It can be used to assess tools that allow the propagation of changes on the right model to the left one in the style of constraint maintainers (where only the result model of the change is known) [7]. Since the right model contains strictly less information than the left one, the dual test case is equivalent to batch forward propagation.

A summary of the features these test cases intend to assess is presented in Fig. 2.

<sup>2</sup>We thus require for this example that propagation be compatible with update composition, i.e., for readers familiar with the delta-lens framework, we demand PUTPUT [3].



Test Cases	Performance		Scalability		Laws			
	Time	Memory	Time	Memory	Correctness	Completeness	Hippocraticness	Least-change
Person2Person.BCF					✓	✓		
Person2Person.BCB					✓	✓		
Person2Person.BSF			✓	✓				
Person2Person.BSB			✓	✓				
Person2Person.MCB					✓	✓	✓	

Figure 2: Feature matrix for the Person2Person BX Benchmark

## 4. RELATED WORK

Benchmarking is an important means of assessing and driving development and improvement in a specific area. In BX related communities, existing benchmarks include for instance [11] for databases, and [12] for graph transformations. These and other existing benchmarks, however, are not directly applicable for BX and cannot be used to address the specific challenges and characteristics of a comparison of BX tools. Our proposal for a BX benchmark format, certainly inspired by existing benchmarks (especially [12]), is explicitly designed to address BX specific aspects.

There also exist tool comparisons and surveys for BX including a quantitative and qualitative comparison of Triple Graph Grammar (TGG) tools by Hildebrandt et. al [5], and the more general survey of BX approaches by Stevens [9]. These survey papers indicate the need for a systematic comparison of existing BX tools but are either too specific (e.g., only covering TGG tools), or too general (no quantitative comparison via a well-defined transformation). Our benchmark proposal is an attempt to fill this gap.

Finally, the Transformation Tool Contest (TTC)<sup>3</sup>, organized yearly, is an ideal venue for presenting challenge transformations and soliciting solutions, which are then compared systematically. Although the TTC 2013 actually had a challenge<sup>4</sup> that required bidirectionality, TTC does not typically focus on BX scenarios. Furthermore, the TTC tends to be specialized for model transformation approaches, while BX encompasses other approaches, e.g., from the programming language or database community.

## 5. DISCUSSION AND CONCLUSION

In this paper, we have identified the properties of a BX benchmark and proposed a format for classifying and presenting BX benchmarks. As a proof of concept, we presented the Persons2Persons BX benchmark as an example that adheres to our format. Note that our example, however, does not fulfil our “usefulness” criterion as it is meant as a minimal template. The next step is to establish a series of BX benchmarks according to the proposed format, extending and refining it as required. There have already been commitments from BX tool developers including Echo,<sup>5</sup> eMoflon,<sup>6</sup> GRoundTram,<sup>7</sup> and HenshinTGG,<sup>8</sup> to provide, support and implement BenchmarX in the near future.

**Acknowledgements.** The authors would like to thank the reviewers for their insightful comments, and all the participants in the 2013 Banff’s meeting on BX that were also involved in the discussion that eventually led to this paper, in particular, Soichiro Hidaka and James Terwilliger.

The second author is funded by ERDF - European Regional De-

<sup>3</sup><http://planet-s1.org/ttc2013/>

<sup>4</sup><http://goo.gl/754XT>

<sup>5</sup><http://haslab.github.io/echo/>

<sup>6</sup><http://www.emoflon.org>

<sup>7</sup><http://www.biglab.org/>

<sup>8</sup><https://github.com/de-tu-berlin-tfs/Henshin-Editor>

velopment Fund through the COMPETE Programme (operational programme for competitiveness) and by national funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FATBIT with reference FCOMP-01-0124-FEDER-020532.

## 6. REFERENCES

- [1] J. Cheney, J. Gibbons, J. McKinna, and P. Stevens. Towards a Repository of BX Examples. In *Proc. of BX 2014*, 2014.
- [2] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. In R. F. Paige, editor, *Proc. of ICMT 2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.
- [3] Z. Diskin, Y. Xiong, and K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *JOT*, 10:1–25, 2011.
- [4] F. Hermann, H. Ehrig, F. Orejas, K. Czarnecki, Z. Diskin, Y. Xiong, S. Gottmann, and T. Engel. Model Synchronization Based on Triple Graph Grammars: Correctness, Completeness and Invertibility. *Software & Systems Modeling*, pages 1–29, 2013.
- [5] S. Hildebrandt, L. Lambers, H. Giese, J. Rieke, J. Greenyer, W. Schäfer, M. Lauder, A. Anjorin, and A. Schürr. A Survey of Triple Graph Grammar Tools. In P. Stevens and J. F. Terwilliger, editors, *Proc. of BX 2013*, volume 57 of *ECEASST*. EASST, 2013.
- [6] M. Hofmann, B. C. Pierce, and D. Wagner. Symmetric Lenses. In T. Ball and M. Sagiv, editors, *Proc. of POPL 2011*, pages 371–384. ACM, 2011.
- [7] L. Meertens. Designing Constraint Maintainers for User Interaction, 1998. Available at <http://www.kestrel.edu/home/people/meertens>.
- [8] A. Schürr and F. Klar. 15 Years of Triple Graph Grammars. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Proc. of ICGT 2008*, volume 5214 of *LNCS*, pages 411–425. Springer, 2008.
- [9] P. Stevens. A Landscape of Bidirectional Model Transformations. In R. Lämmel, J. Visser, and J. a. Saraiva, editors, *Proc. of GTTSE 2008*, volume 5235 of *LNCS*, pages 408–424. Springer, 2008.
- [10] P. Stevens. Towards an Algebraic Theory of Bidirectional Transformations. In H. Ehrig, R. Heckel, G. Rozenberg, and G. Taentzer, editors, *Proc. of ICGT 2008*, volume 5214 of *LNCS*, pages 1–17. Springer, 2008.
- [11] Transaction Processing Performance Council. TPC Benchmark C (Standard Specification, Revision 5.11), 2010. <http://www.tpc.org/tpcc/>.
- [12] G. Varró, A. Schürr, and D. Varró. Benchmarking for Graph Transformation. In *Proc. of VL/HCC 2005*, pages 79–88, 2005.

# Towards a Repository of Bx Examples

James Cheney, James McKinna,  
Perdita Stevens  
School of Informatics  
University of Edinburgh  
firstname.lastname@ed.ac.uk

Jeremy Gibbons  
Department of Computer Science  
University of Oxford  
firstname.lastname@cs.ox.ac.uk

## ABSTRACT

We argue for the creation of a curated repository of examples of bidirectional transformations (bx). In particular, such a resource may support research on bx, especially cross-fertilisation between the different communities involved. We have initiated a bx repository, which is introduced in this paper. We discuss our design decisions and their rationale, and illustrate them using the now classic *Composers* example. We discuss the difficulties that this undertaking may face, and comment on how they may be overcome.

## 1. INTRODUCTION

Research into bidirectional transformations (henceforth: **bx**) involves a wide range of disciplines, from databases, to model-driven development, to programming languages. The literature is rich in examples illustrating the corresponding wealth of notations and tools used to formalise the variety of bx as they occur “in the wild”.

In writing a paper about bidirectional transformations, one typically needs to use examples. Several examples, such as the notorious UML class diagram to RDBMS schema example, have appeared in many variants in papers by many authors. It can be difficult to identify whether examples in different papers are really identical; moreover, the need to fully define every example mentioned often makes it difficult to use more than one or two examples in a paper that is subject to a page limit, and can lead to examples being defined so concisely as to make it difficult for a reader to understand them. This, in turn, leads to proliferation of versions.

In 2013 at the ETAPS Bx workshop<sup>1</sup>, we announced the development of a repository of examples, to be made publicly available on the Bx wiki [14]. We have a personal interest in such a repository, as a foundation for our own work in the EPSRC-funded project *A Theory of Least Change for Bidirectional Transformations*; but we hope that it will have much broader use. The repository aims to simplify reuse of examples, especially in order that meaningful comparisons between formalisms will be easier to make. It aims to improve communication between sometimes quite disparate communities, and to help explain bx to interested outsiders. We hope

<sup>1</sup><http://bx-community.wikidot.com/bx2013:home>

that researchers in bx will both add their examples to the repository, and refer to examples from the repository as appropriate. This should have benefits both for authors and for readers. Naturally, papers will still have to be sufficiently self-contained; but we think that the repository may still be helpful. For example, if one’s paper illustrates some new work using an existing example, one might describe the example briefly (as at present), focusing on the aspects most relevant to the work at hand. One can, however, also give a reference into the repository, where there is more detail and discussion of the example, which some readers may find helpful. Over time, we might expect that certain examples in the repository become familiar to most researchers in bx. Then, if a paper says it uses such an example, the reader’s task is eased. The repository should also be a good place to link in auxiliary materials, such as files showing how an example is implemented in different formalisms, or diagrams suitable for inclusion in papers and talks.

Repositories of examples are not easy to make successful, however; if this one is to succeed, it is important to design its formats and processes with care, and to engage the community. This paper describes our initial ideas and the rationale behind them.

Concretely, we illustrate what we have in mind through the *Composers* example. We do not intend that every example in the repository will follow the format illustrated here – different examples may have different needs. Our intention is that our template will at least provide a basis for discussion for future iterations. The repository of examples is hosted on the Bx wiki at:  
<http://bx-community.wikidot.com/examples:home>

## 2. DESIGN AND RATIONALE

We draw some inspiration from the design patterns movement [6], but not so much in the sense of ‘a catalogue of expert advice for assisting novices’. Indeed, there are socio-technical arguments as to why this is a rather difficult case to make [5]. An easier case to make for the benefits of patterns is that they improve communication between already proficient practitioners; and it is this analogy that we hope to make in considering the design of the repository.

One way in which design patterns improve communication is simply by capturing and recording ‘folk knowledge’ – the concepts and stories known and shared by everyone in a community, and assumed to be so known, so not always explained in detail. By explicitly recording our corpus of common examples, we remind ourselves of their existence, and we provide a resource for filling in the inevitable gaps in our coverage of this ‘folk knowledge’.

A second way in which design patterns improve communication is by providing a common vocabulary. The naming of entries is a difficult matter; but a well-chosen name takes its place in the community’s discourse, evoking a concept or a story and invoking its connotations. But for the purposes of scientific communication,

the name itself is not enough. We need also to be able to maintain a stable reference for each example in the repository, so that it can be referenced in a paper with some hope that that reference will persist. We also need to have some confidence that example descriptions themselves are stable; so we expect to have to curate the repository, encouraging free discussion and commentary but versioning the descriptions at appropriate points.

Also extrapolating from the history of design patterns, we suggest that there is value in having some regularity of the structure between examples in the repository. We propose a standard template for our examples, which we discuss in more detail below. We do not intend to be too prescriptive and rigid about this template; for one thing, we expect that our understanding of the best structure will evolve with experience, and for a second, examples arising from different parts of the bx community, or examples that differ in character, may well warrant different structures.

It should be noted that one way in which the example repository diverges from the patterns movement is that the entries in the repository do not follow Alexander's three-part rule of context, problem, and solution – there are no conflicting forces to resolve, nor necessarily any expectation of a resolution. Rather, if the examples are to be considered ‘problems’ at all, then it is in the Kuhnian sense of the problems and techniques that form a paradigm of ‘normal science’ [9]. Nevertheless, we intend that the repository will provide access to artefacts as well as descriptions, be they executable code, proof scripts, sample inputs and outputs, or what have you.

We wish to maintain a “broad church” approach to what can be included in the repository. We think that the most useful entries will be small ones that can be defined precisely, but in a way that is as independent as possible of any particular bx formalism. These we intend to describe principally in English (perhaps augmented with small amounts of simple mathematical notation) but with sufficient precision that readers familiar with a particular formalism should be able to tell, unambiguously, whether a representation of a bx in that language is correctly implementing the example. The review process, to be discussed, should play an important role in eliminating ambiguities. Artefacts showing how examples are implemented in particular formalisms may also be helpful.

Besides these, several other classes of examples may be anticipated. We agree with the authors of [1] (in this volume) that benchmarks may be seen as a distinct class and therefore should be included. Industrial-scale examples, accompanied by appropriate artefacts, would clearly be of interest, but equally clearly, could not be expected to be explained with full precision separately from their artefacts. Other examples we have in mind are more like sketches: situations in which a certain bx would clearly have applicability, but where details have not been worked out. These might be of particular benefit to outsiders wondering whether bx are of interest to them. We have adopted the suggestion made in group discussion at the Bx workshop in Banff in December 2013<sup>2</sup>, namely to make explicit the class to which an example belongs, because these classes may be quite different in character.

### 3. A TEMPLATE FOR BX EXAMPLES

We propose the following template for examples. It consists of the following headings, whose kernel is the description of bx given, for example, by Stevens in [13]. That is, an example will typically define two or more classes of models, together with a consistency relation between them, and appropriate consistency restoration functions. Such functions might depend just on the states of

<sup>2</sup><http://www.birs.ca/events/2013/5-day-workshops/13w5115>

the models (state-based bx) or might require as input extra information, e.g. concerning the edit that has been done. Either is fine provided it is clear what is assumed.

We propose the following standard fields and their order. Optional fields are indicated by ‘?’ in the fieldname; other fields should be present, even if brief.

**Title** A descriptive name, such as COMPOSERS, by which authors may refer to the example. More advanced indexing, and traceability back to the originating sources (see under **References** below) may prove necessary as the repository grows.

**Version** 0.x for unreviewed examples.

**Type** For example, PRECISE, INDUSTRIAL, SKETCH. Use common sense concerning whether to use one or more: for example, PRECISE and SKETCH should be mutually exclusive, but conceivably either might be combined with INDUSTRIAL.

**Overview** A thumbnail description of the example, not more than two or three sentences; might include a brief summary of the example and/or a brief reason for its inclusion in the repository (“demonstrates [some interesting point]” for example).

**Models** Descriptions of the models, possibly with (formal) expressions of their meta-models. (We remind readers that we use the term “model”, and “meta-model” inclusively: any appropriately precise description of the information sources being transformed is acceptable.)

**Consistency** Description of the consistency relationship between models. This should at least be in natural language, but may be augmented by formal expression in some language cognate with that of the meta-models.

**Consistency Restoration** Explain in which of the typically many possible ways inconsistencies are to be repaired. May be divided into separate descriptions of forward and backward restoration.

**Properties?** What additional properties are expected to hold of, or be exemplified by, the transformation? These will link to a separate glossary of terms such as ‘hippocraticness’.

**Variants?** A difficult issue that we have found arises in writing examples is how to handle the choice points. Typically in making an example precise, even a small one, one realises that there are several points where more than one choice is reasonable. These multiply to give a potentially unmanageable set of examples. Our proposal is that one “base” example should be given in the main body of the text, and variation points be described here.

**Discussion** Origin, utility, interest, representativeness, related examples in the literature, . . .

**References?** Bibliographic data for the paper or papers from which the example is taken, or where it is discussed, if applicable.

**Authors** Contributing author(s) of the example to the repository.

**Reviewers?** We intend that examples remain provisional (version 0.x) until reviewed (and approved, if necessary after modification) by other members of the wiki. In the interest of traceability and credit, such reviewers are identified here.

**Comments** This is where any member of the wiki can comment. As discussed later, we do not wish to have uncontrolled editing of the example itself, but it is important to make it easy for community members to make comments. These might guide the development of a later version of the example.

**Artefacts?** Formal descriptions, perhaps downloadable code, sample input and output, virtual machine instances, diagrams...

## 4. AN EXAMPLE INSTANCE

We choose to illustrate our ideas with the familiar example of *Composers*, which first appeared in [3] and has been used by several others since. This version is closest to the one in [12].

**Title** COMPOSERS

**Version** 0.1

**Type** PRECISE

**Overview** This example stands for many cases where two slightly, but significantly, different representations of the same real world data are needed. The definition of consistency is easy, but there is a choice of ways to restore consistency.

**Models** A model  $m \in M$  comprises a set of (unrelated) objects of class *Composer*, representing musical composers, each with a name, dates and nationality.

A model  $n \in N$  is an ordered list of pairs, each comprising a name and a nationality.

**Consistency** Models  $m$  and  $n$  are consistent if they embody the same *set* of (name, nationality) pairs. That is, both: (i) for every composer in  $m$ , there is at least one entry in the list  $n$  with the same name and nationality; and (ii) for every entry in  $n$ , there is at least one element of  $m$  with the same name and nationality (there may be many such, each with distinct dates).

**Consistency Restoration** Given models  $m$  and  $n$ ,

**Forward** produce a modified version of  $n$  by:

- deleting from  $n$  any entry for which there is no element of  $m$  with the same name and nationality;
- adding at the end of  $n$  an entry comprising each (name, nationality) pair derivable from an element of  $m$  but not already occurring in  $n$ . Such additional entries should be in alphabetical order by name, and within name, by nationality; no duplicates should be added (even if there are several composers in  $m$  with the same name and nationality).

**Backward** produce a modified version of  $m$  by:

- deleting from  $m$  any composer for which there is no entry in  $n$  with the same name and nationality;
- adding to  $m$  a new composer for each (name, nationality) pair that occurs in  $n$  but is not derivable from an element already occurring in  $m$ . The dates of any newly added composer should be ???-???

**Properties**

- Correct
- Hippocratic

- Not undoable
- Simply matching

**Variants** If the  $bx$  is to be correct and hippocratic, restoring consistency must involve adding (respectively, deleting) composers that are present (respectively, not present) in the authoritative model but not in the one to be modified. Questions that the  $bx$  programmer still needs to resolve are:

- Do we ever modify the name and/or nationality of an existing composer, or do we create a new composer in the event of any mismatch? E.g. if one side has *Britten, British* and the other has *Britten, English*, does consistency restoration involve changing one of the nationalities, or adding a second Britten? Of course, if name is a key in the models then there is no choice.
- Where in the list  $n$  is a new composer added? Choices include: at the beginning; at the end. We might consider in an alphabetically determined position, but note that the user is not constrained to add composers in alphabetical order, and we fail hippocraticness if we choose to reorder when nothing at all need be changed. It therefore seems unlikely that changing the order of user-added composers will be wanted.
- What dates are used for a newly added composer in  $m$ ?

**Discussion** This has been used as an example of why undoability is too strong. Consider a composer currently present (just once) in both of a consistent pair of models. If we delete it from  $n$ , and enforce consistency on  $m$ , the representation of the composer in  $m$ , including this composer's dates, is lost. If we now restore it to  $n$  and re-enforce consistency on  $m$ , then the absence of any extra information besides the models means that the dates cannot be restored, so  $m$  cannot return to exactly its original state.

**References** This version was used in:

Perdita Stevens, "A Landscape of Bidirectional Model Transformations", in *Generative and Transformational Techniques in Software Engineering II*, 2008, Springer LNCS 5235, pp408–424. DOI 10.1007/978-3-540-75209-7\_1

Original (asymmetric) variant was in:

Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. "Boomerang: Resourceful Lenses for String Data". In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, San Francisco, California, January 2008. DOI 10.1145/1328438.1328487

**Author(s)** Perdita Stevens, James McKinna, James Cheney. University of Edinburgh.

**Reviewer(s)** None yet

**Comments** None yet

## 5. DIFFICULTIES AND HOW THEY MAY BE TACKLED

We are only too aware that it is far easier to found a repository such as proposed here than it is to ensure its success. In this section we attempt to address some questions that a sceptical reader might ask.

## 5.1 Can we ensure and maintain the quality of the repository contents?

A key decision we have made is that the example repository will be a *curated* resource in the sense of Buneman *et al.* [4]. This means that it is put together by human effort by members of a knowledgeable community – as opposed, say, to being extracted automatically from a web crawl. Other examples include many databases constructed to cover various areas of biomedical research (sometimes using wikis, sometimes using custom-developed websites). Example repositories in software engineering, such as ReMoDD [2], SPLOT [11], 101companies [10], and the Hillside patterns wiki [7], could also be viewed as curated in this sense; notice that there is a wide range of curation styles. One can also view Wikipedia as an example of this phenomenon, and some biological wikis are hosted there. These efforts have common technical and organizational structure (and encounter similar problems) and our aim is to learn from them.

Resources such as Wikipedia can be edited by anyone with an Internet connection. Unhelpful contributions are common, with editors and bots reverting such changes frequently. By contrast, resources curated by smaller, focused communities typically do not have this problem, especially if there is a barrier to entry, such as registration to obtain a wiki account. This is the model we plan to follow.

Specifically, we propose a three-level curatorial structure for the repository. Anyone with a wiki account will be able to comment on an example, and this should enable anybody to point out problems, ambiguities, extensions and any other points of interest. Eventually, each example will also have one or more named reviewers: recognised members of the community whose name as reviewer indicates they consider the example to be of usable quality. Overall editorial control of the repository is the responsibility of a small group of curators, initially ourselves.

At a technical level, there is a tension between fixing a structure for the curated data (such as our proposed example template) and allowing free-text or lightweight markup only. Both fixing a template and having no template at all can lead to headaches later; the former risks being too rigid for the application and forcing a premature decision about what the best template is, while the latter diminishes the integrity and intelligibility of the resource. We aim to take a middle road, providing a suggested template but not a barrier to varying it where good reasons to do so arise. In the longer term, adopting a more structured solution (e.g. to facilitate a move to a different platform than a wiki) may be justifiable.

## 5.2 Will anybody use it?

It has frequently been observed (*e.g.* by Josuttis in [8]) that companies that attempt to increase software reuse by creating a library of reusable components frequently find it easier to get people to put things into the library than to take them out.

In this case, informal discussion of the idea at two Bx meetings (in Rome and Banff) together with separate discussions with individuals suggest that there is a genuine need for examples and a willingness to refer to examples in a repository.

To help maximise the chance that the repository is used, we will seek to:

- host the repository on the main long-lived community site, the Bx wiki;
- use a format that makes it easy to find information quickly;
- provide guidance on how to refer to examples;

- keep old versions of examples available, so that old references can still be followed;
- introduce a reviewing mechanism to help ensure high quality of the examples.

To give more detail, we may split this question into several parts.

### *Will relevant people know about the repository?*

Our aim in talking about our plans even before the repository existed was not only to check that we were targeting a genuine community need, but also to spread the word. The present paper is an attempt to give more detail. Further, by hosting the repository on the main Bx community site, the Bx wiki, we hope it will be easy to find.

### *Will people be able to find and refer to relevant examples?*

Suppose that we have succeeded in answering the earlier question positively so that the repository does contain relevant, high quality examples: this is still no use unless people can find and use them. We might worry about a future in which there are so many examples in the repository that it is hard to find the right one; but experience suggests that ensuring that the wiki is (as at present) google indexed goes a long way to solving this problem.

In this specific case, another important aspect of use is the ability to refer accurately to an example, for example, in a paper that uses it. Readers seeing the reference need to be able to identify exactly the example referred to.

As already discussed, well-chosen names are important for the usability of our examples. However, versioning and variability management are likely to be problems; we have discussed this issue among ourselves and it has been raised by others at the Banff workshop also. It is important to distinguish between *versioning* and *variation*. Both require assigning distinct identifiers, but versioning connotes a (sequential) evolution of a single example, while variation connotes related variants of similar examples.

We could devise a system in which a short identifier represented the example, its version and the choice at each variation point. However, in the absence of automated support for such a system, we believe that taking care to choose example names carefully, to identify a single main variation within each example, and to maintain a linear sequence of numbered versions, will be more usable, and adequate for the time being.

### *Will anybody but ourselves put any examples in?*

Experience with curated databases and crowdsourcing projects suggests that it is very important to pay attention to incentives for contributing (and maintaining) quality examples. We have already noted the importance of versioning; in addition to providing unique identifiers, it seems like a good idea to recommend a format for citations to examples (including versions) or to the repository itself. In addition, we hope that listing authors and reviewers of examples will incentivise contributions of examples and of effort to review them. If the repository reaches a point of relative maturity or stability, it may make sense to collect the most recent versions of all of the examples in it into a manuscript (with all authors and reviewers named), and publish it formally as a citable, archival technical report.

## 5.3 Will the contents rot?

For the next three years, we the authors can undertake to curate the repository under the aegis of the Least Change research project.

By that time, we hope that it will have gathered enough momentum that if necessary, finding other volunteer curators may be possible.

## 5.4 What happens if the Bx wiki dies?

We hope that the Bx wiki will flourish. We shall, however, maintain a local copy of the repository contents, in case of future difficulties. We plan to give some thought to whether maintaining it in a wiki-markup-independent form, and maintaining consistency between that and the wiki via a bidirectional transformation, might add value.

## 6. CONCLUSION

In the spirit of earlier documentary movements in the software engineering literature, such as that of the Patterns community, we have articulated the rationale for a repository of example bx, as well as providing here a canonical reference in the literature to its existence.

We hope that colleagues in the Bx community will wish to contribute to the repository, to use it in their own work and papers, and to discuss with us improved versions of the template described here. An early example is discussion that has just begun with authors of [1] about extra optional sections that may be necessary for benchmark examples.

## 7. ACKNOWLEDGEMENTS

We gratefully acknowledge the contributions made by participants of the two 2013 Bx workshops, in Rome and Banff, and by the anonymous reviewers. The work reported here was supported by the EPSRC-funded project *A Theory of Least Change for Bidirectional Transformations* (EP/K020218/1 and EP/K020919/1).

## 8. REFERENCES

- [1] Anthony Anjorin, Manuel Alcino Cunha, Holger Giese, Frank Hermann, Arend Rensink, and Andy Schürr. BenchmarkX. In *Proceedings of Bx'14*, 2014.
- [2] James M. Bieman, Betty H. C. Cheng, and Robert B. France. ReMoDD: The repository for model-driven development. <http://www.cs.colostate.edu/remodd/>.
- [3] Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz, and Alan Schmitt. Boomerang: Resourceful lenses for string data. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, San Francisco, California, January 2008.
- [4] Peter Buneman, James Cheney, Wang-Chiew Tan, and Stijn Vansummeren. Curated databases. In *Proceedings of the 2008 Symposium on Principles of Database Systems (PODS 2008)*, pages 1–12, 2008. Invited paper.
- [5] Luciana d’Adderio, Rick Dewar, Ashley Lloyd, and Perdita Stevens. Has the pattern emperor any clothes? A controversy in three acts. *ACM SIGSOFT Software Engineering Notes*, Jan/Feb 2002. <http://dx.doi.org/10.1145/566493.1148026>.
- [6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [7] Hillside Group. The Hillside patterns wiki. <http://c2.com/cgi/wiki>.
- [8] Nicolai M. Josuttis. *SOA in Practice: The Art of Distributed System Design*. O’Reilly, 2007. <http://www.soa-in-practice.com/>.
- [9] Thomas S. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, 1962.
- [10] Ralf Lämmel. The 101companies project. <http://101companies.org/>.
- [11] Marcilio Mendonca. SPLIT: Software product lines online tools. <http://www.splot-research.org/>.
- [12] Perdita Stevens. A landscape of bidirectional model transformations. In Ralf Lämmel, Joost Visser, and João Saraiva, editors, *Generative and Transformational Techniques in Software Engineering (GTTSE)*, volume 5235 of *Lecture Notes in Computer Science*, pages 408–424. Springer, 2007.
- [13] Perdita Stevens. Bidirectional model transformations in QVT: Semantic issues and open questions. *Journal of Software and Systems Modeling (SoSyM)*, 9(1):7–20, 2010.
- [14] The Bx Community. Bidirectional transformations: The Bx wiki. <http://bx-community.wikidot.com/>.

# Intersection Schemas as a Dataspace Integration Technique

Richard Brownlow  
Department of Computer Science  
Birkbeck, University of London  
London, UK  
richard@dcs.bbk.ac.uk

Alex Poulouvassilis  
Department of Computer Science  
Birkbeck, University of London  
London, UK  
ap@dcs.bbk.ac.uk

## ABSTRACT

This paper introduces the concept of *Intersection Schemas* in the field of heterogeneous data integration and dataspace. We introduce a technique for incrementally integrating heterogeneous data sources by specifying semantic overlaps between sets of extensional schemas using bidirectional schema transformations, and automatically combining them into a global schema at each iteration of the integration process. We propose an incremental data integration methodology that uses this technique and that aims to reduce the amount of up-front effort required. Such approaches to data integration are often described as *pay-as-you-go*. A demonstrator of our technique is described, which utilizes a new graphical user tool implemented using the AutoMed heterogeneous data integration system. A case study is also described, and our technique and integration methodology are compared with a classical data integration strategy.

## Keywords

Dataspace, Pay-as-you-go, Data Integration, Bidirectional Schema Transformations

## 1. INTRODUCTION

Data integration is the process of taking data from several different data sources and bringing them together in a structured manner such that Data Services can be supported over the integrated resource. These data services could be Data Mining tools, search engines or simple querying tools. The data sources can be varied in type, for example relational databases, XML data, or web pages. The data sources might be located at different sites of a network, and each data source may be running different data management software. The overarching challenge in data integration is to design frameworks and methodologies that allow heterogeneous data sources to be accessed as a single integrated resource by data services.

Traditional data integration approaches [5] require all the mappings between the different data sources to be determined up-front, prior to being able to run any data services

on the integrated resource. In order to establish the mappings, it is likely a data integration project will require a certain level of domain expertise. This approach to data integration means that the full cost of the integration must be committed up front. In order to do this accurately, the timescales for producing the mappings must be accurately estimated. As a result, data integration projects are often costly and risky undertakings [9].

An active area of research [8, 9] is how to reduce the amount of up front effort required in data integration. One approach is to develop a framework that presents all the data in a Common Data Model, but in an unintegrated format. Tools are then provided to allow the data integrator to incrementally identify the semantic relationships between the different data sources. The concept of a dataspace allows semantic integration of data to be undertaken incrementally [7]. Such an approach is often described as *pay-as-you-go*, since integration can proceed as resources (and budgets) allow. Data services can then be provided at each iteration, rather than waiting for all the integration to be completed up-front.

A theoretical discussion of data integration is presented in [12]. This includes a discussion of global-as-view (GAV) and local-as-view (LAV) approaches, along with a discussion of query processing in data integration settings. The data integration setting is formally defined in terms source schemas, global schemas and mappings between source and global schemas.

The data integration process can be considered as comprising three subprocesses. Firstly, schema matching and mapping, which includes the identification of correspondences between different schema objects and the definition of mappings between schemas. The work in [17] provides an overview of the schema matching process, while [3, 1, 4] discuss different approaches to schema mapping. Secondly, schema merging — the creation of an integrated schema based on the schema mappings. Thirdly, schema improvement which increases the quality of the integrated schema, for example by removing redundant information. The work in [2] discusses the schema improvement and refinement process. An incremental (or *pay-as-you-go*) data integration process results in an incomplete integration after each iteration of these three subprocesses.

Data integrators tend to fall into two groups [9]. The first

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

group are domain experts who are knowledgeable in the application domain(s) of the data sources being integrated (often with limited data integration experience). The second group are data integration experts who are familiar with data integration environments (but are likely to have limited experience in the target application domain for a particular data integration project). These two groups of integrators must work closely together. It is therefore important to develop tools which can be used by both of these types of data integrators, to increase their productivity in the data integration environment, while still allowing maximum flexibility in the types of semantic mappings that can be supported.

This paper proposes a new technique for reducing the amount of up front effort required in data integration, utilizing bidirectional schema transformations. The focus of our research is developing *light-weight* techniques for creating mappings between data sources and in developing new data integration frameworks and methodologies that use such techniques. By *light-weight* techniques we mean techniques that can be used for rapid, incremental dataspace integration. In Section 2 of the paper we present in detail our approach. We present an overview of the AutoMed system in Section 2.1, our new intersection schema methodology in Section 2.2, a description of the methodology integration workflow in Section 2.3 and a worked example in Section 2.4. In Section 3 we present a case study contrasting our approach with a classical data integration approach in the context of a large-scale data integration project in the field of proteomics. We give our concluding remarks and directions of further work in Section 4.

## 2. OUR APPROACH

### 2.1 Overview of AutoMed

AutoMed<sup>1</sup> is a schema transformation and integration system that provides a low-level hypergraph-based data model (the *HDM*) as a common data model for representing heterogeneous data sources. Higher level modelling languages (e.g. relational, XML, RDF/S, OWL) can be defined in terms of the HDM using the API of AutoMed’s Model Definitions Repository (MDR) [6].

For any modelling language  $M$  specified in terms of the HDM, AutoMed provides a set of primitive schema transformations that can be applied to schema constructs expressed in  $M$ . In particular, there is an *add* and a *delete* primitive transformation for adding/deleting any construct of  $M$  to/from a schema. For those constructs of  $M$  which have textual names, there is also a *rename* primitive transformation. Each *add* or *delete* transformation is accompanied by a query which defines the extent of the new or deleted schema object in terms of the rest of the objects in the schema (i.e. this query specifies a view definition). This query is expressed in IQL, a functional query language developed for the AutoMed system [10]. Also available are *extend* and *contract* primitive transformations which behave in the same way as *add* and *delete* except that they state that the extent of the new/deleted schema object cannot be precisely derived from the other schema objects. Each *extend* and *contract* transformation takes a query of the form *Range*  $q_l$   $q_u$  where the subqueries  $q_l$  and  $q_u$  specify a lower and an upper

bound on the extent of the new/deleted schema object. The lower bound may be the constant *Void*, equivalent to the empty collection, and the upper bound may be the constant *Any*, equivalent to the largest possible collection of the type of the schema object.

Schemas and their associated instances can be incrementally transformed by applying to them a sequence of primitive transformations. A sequence of primitive transformations transforming a schema  $S_1$  to a schema  $S_2$  is termed a *pathway* from  $S_1$  to  $S_2$  and denoted by  $S_1 \rightarrow S_2$ . All source, intermediate and integrated schemas, and the pathways between them, are stored in AutoMed’s Schemas & Transformations Repository (STR) [6].

In addition to the five primitive schema transformations already mentioned, AutoMed also supports an *ident* primitive. This operates at the level of entire schemas allows the integrator to assert that two syntactically identical schemas,  $S$  and  $S'$ , should be connected by a pathway. The AutoMed system automatically translates such an *ident* transformation into a sequence of *id* transformations from  $S$  to  $S'$ , of the form *id*( $S : c, S' : c$ ) for each schema object  $c$  appearing in  $S$  and  $S'$  (where  $S : c$  denotes object  $c$  appearing in schema  $S$ ).

A key property of AutoMed’s pathways is that they are automatically reversible, in that from a pathway  $S_1 \rightarrow S_2$  we can automatically derive a pathway  $S_2 \rightarrow S_1$  by reversing the order of the transformations and replacing each *add* by a *delete* with the same arguments, and vice versa; each *extend* by a *contract* with the same arguments, and vice versa; and each *rename* or *id* by a *rename* or *id* with the arguments reversed.

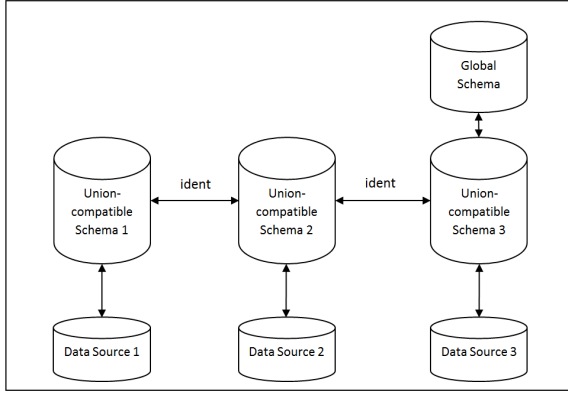
Schema integration using AutoMed typically proceeds by forming *union-compatible* schemas, as illustrated in Figure 1. Firstly, the appropriate AutoMed Wrapper for each data source is used to extract metadata from the data sources and to produce a set of data source schemas,  $DS_1, \dots, DS_n$ , stored within the AutoMed repository. In order to integrate these, each  $DS_i$  is first transformed into a union-compatible schema  $US_i$ . The  $n$  schemas  $US_1, \dots, US_n$  are identical, and this is verified by injecting an *ident* transformation between each pair  $US_i$  and  $US_{i+1}$ . An arbitrary one of the  $US_i$  can then be selected by the user for further improvement and refinement into the global schema. In terms of the data extents of the objects in the global schema, AutoMed provides a number of options for deriving these from the extents of objects in the union-compatible schemas, e.g. bag union, set union, choice, intersection. The first of these is the default, and it is what we assume in this paper.

There may be information within a  $US_i$  which is not semantically derivable from the corresponding  $DS_i$ . This is indicated by *extend* transformation steps within the pathway  $DS_i \rightarrow US_i$ . Conversely, not all of the information within a data source schema  $DS_i$  need be transferred into  $US_i$  and this is indicated by *contract* transformation steps occurring within  $DS_i \rightarrow US_i$ .

The queries accompanying the primitive transformations are used by AutoMed’s Query Processor [10] in order to refor-

<sup>1</sup><http://www.doc.ic.ac.uk/automed>





**Figure 1: Data Integration via Union-Compatible Schemas**

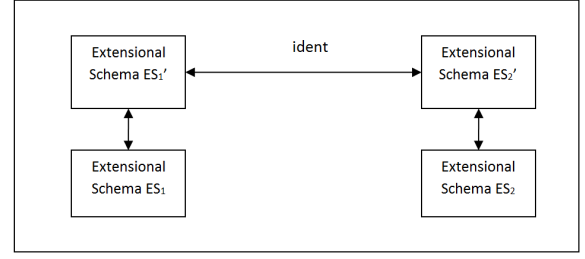
mulate users’ queries expressed on a global schema to queries expressed on the source schemas. Query reformulation may be by means of query unfolding for GAV query processing, or by rewriting queries using views for LAV query processing [12], or by a combination of both GAV and LAV query processing techniques [15]. In [14], AutoMed’s transformation approach was termed Both-As-View (BAV) since the add/extend steps in a pathway correspond to GAV mappings and the delete/contract steps to LAV mappings. However, BAV transformations capture a finer level of data integration granularity than do conventional GAV or LAV mappings, since BAV transformations are stated on the irreducible modelling constructs of a modelling language  $M$  (as determined by the definition of the language  $M$  in terms of the HDM), e.g. on individual columns of relational tables as opposed to on whole tables. It is also possible to express global-local-as-view (GLAV) mappings using BAV transformations [11].

A typical data integration workflow using the AutoMed system proceeds as follows: Firstly, each data source is wrapped to produce a data source schema. Secondly, the schema matching and mapping process is undertaken, with the help of AutoMed’s Schema Matching tool [16] and the knowledge of domain experts. Each data source schema is transformed to a union-compatible schema via a series of primitive transformations. Thirdly, the union-compatible schemas are automatically merged by injecting *ident* transformations between them. Finally, one of these schemas is chosen as the source for further transformation, capturing any necessary improvements and refinements into the final global schema.

## 2.2 Intersection Schemas

We now propose a new methodology for lightweight data integration in an incremental pay-as-you-go environment, based on the concept of *Intersection Schemas*. The primary goal of this approach is to improve on existing data integration methodologies by increasing data integrators’ productivity in the overall Data Integration process.

We demonstrate a lightweight technique that allows a domain expert to identify schema mappings that represent semantic intersections between different data source schemas. Using our technique, both the schema matching/mapping



**Figure 2: Intersection Schema**

and the schema merging processes are undertaken iteratively.

Three types of schema are encompassed within our methodology:

*Extensional Schema:* This is any schema in the AutoMed repository that is connected to a data source schema via a pathway.

*Federated Schema:* This is a combination of multiple schemas (of any kind),  $S_1, \dots, S_n$ , into a single virtual schema  $F$  containing a union of the objects in  $S_1, \dots, S_n$ , without undertaking any schema or data transformation or integration. Within  $F$ , the schema objects from each  $S_i$  are prefixed with the schema identifier of  $S_i$  so as to (i) be able to easily distinguish their provenance and (ii) disambiguate objects of the same name from different schemas. We write:

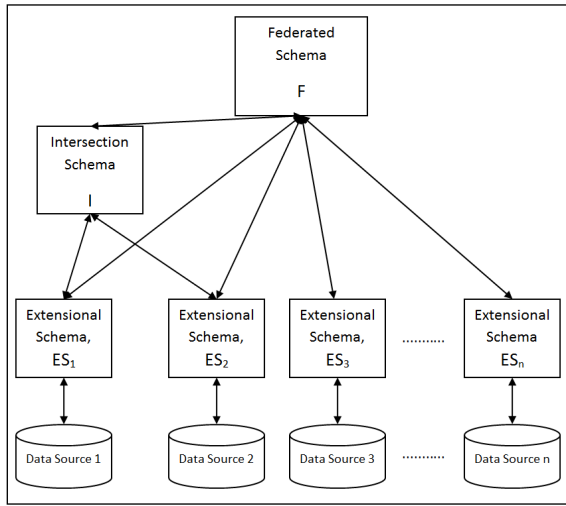
$$F = S_1 \cup S_2 \cup \dots \cup S_n$$

*Intersection Schema:* This is a schema which contains only semantically overlapping content from a pair of extensional schemas  $ES_1$  and  $ES_2$  (see Figure 2). In more detail, there need to exist in the AutoMed repository two pathways  $ES_1 \rightarrow ES_1'$  and  $ES_2 \rightarrow ES_2'$  with  $ES_1'$  and  $ES_2'$  being union-compatible schemas, and each pathway consisting of a sequence of *add* and *delete* operations followed by a sequence of *contract* operations. Specifically, the pathway  $ES_1 \rightarrow ES_1'$  will be of the form:

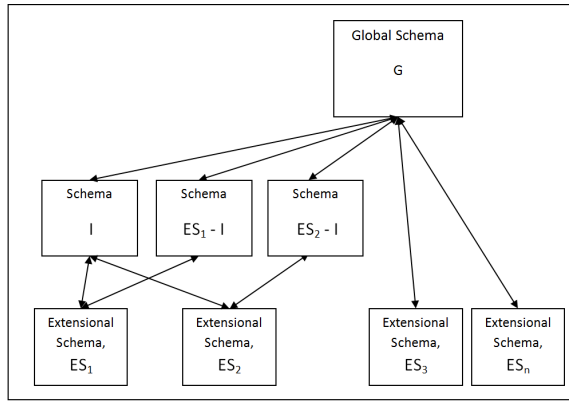
$$\begin{aligned} &add(o_1, f_1(ES_1)), add(o_2, f_2(ES_1)), \dots, add(o_r, f_r(ES_1)), \\ &del(c_1, g_1(ES_1')), del(c_2, g_2(ES_1')), \dots, del(c_m, g_m(ES_1')), \\ &contract(c_{m+1}, Range Void Any), \\ &contract(c_{m+2}, Range Void Any), \\ &contract(c_n, Range Void Any) \end{aligned}$$

where the  $c_i$  are schema objects of  $ES_1$ , the  $o_i$  are schema objects of  $ES_1'$ , the  $f_i$  are IQL queries over  $ES_1$  and the  $g_i$  are IQL queries over  $ES_1'$  (the pathway  $ES_2 \rightarrow ES_2'$  has a similar form). The part of this pathway comprising the *add* and *delete* steps asserts the semantic equivalence of the set of schema objects  $o_1, \dots, o_r$  of  $ES_1'$  and the set of objects  $c_1 \dots c_m$  of  $ES_1$ . There is also a pathway  $ES_1' \rightarrow ES_2'$  comprising a single *ident* operation.

The schemas  $ES_1'$  and  $ES_2'$  can both be regarded as intersection schemas, and one of them can be explicitly renamed to reflect the fact that it is specifically chosen to be the intersection schema,  $I$ . As noted in Section 2.1, the data extents of the objects in schema  $I$  are formed from a bag-union of



**Figure 3: Integrated Intersection and Extensional Schemas**



**Figure 4: Global schema derived from Intersection and Extensional Schemas**

the objects in schemas  $ES'_1$  and  $ES'_2$  from which they are derived.

In our approach a federated schema can be created from a set of extensional schemas  $ES_1, \dots, ES_n$  and a set of intersection schemas  $I_1, \dots, I_m$  (see Figure 3):

$$F = ES_1 \cup \dots \cup ES_n \cup I_1 \cup \dots \cup I_m$$

If the intersection schemas  $I_1, \dots, I_m$  have been derived from a subset of the extensional schemas  $ES_1, \dots, ES_n$  then it is possible to automatically determine objects that are now semantically redundant in  $F$  and our implemented tool (see Section 2.3 below) offers this as an option. For example, in Figure 4, the intersection schema  $I$  has been derived from the extensional schemas  $ES_1$  and  $ES_2$ , and the global schema  $G$  is defined as:

$$G = I \cup (ES_1 - I) \cup (ES_2 - I) \cup ES_3 \dots \cup ES_n$$

The  $-$  operator here removes from the schema that is its first argument the subset of objects that are semantically equiv-

alent with some subset of the objects in the schema that is its second argument. Operationally, given two extensional schemas  $ES_1$  and  $ES_2$  and an intersection schema  $I$  derived from them as described earlier, the schema  $ES_1 - I$  is obtained from  $ES_1$  by retaining only those schema objects of  $ES_1$  which have been removed in the pathway  $ES_1 \rightarrow I$  by a *contract* operation; likewise for the schema  $ES_2 - I$ . In more detail, if the pathway  $ES_1 \rightarrow I$  is of the following form (possibly with a sequence of *ident* transformations added at the end as well):

$$\begin{aligned} &add(o_1, f_1(ES_1)), add(o_2, f_2(ES_1)), \dots, add(o_r, f_r(ES_1)), \\ &del(c_1, g_1(I)), del(c_2, g_2(I)), \dots, del(c_m, g_m(I)), \\ &contract(c_{m+1}, Range Void Any), \\ &contract(c_{m+2}, Range Void Any), \\ &contract(c_n, Range Void Any) \end{aligned}$$

where the  $c_i$  are schema objects of  $ES_1$ , the  $o_i$  are schema objects of  $I$ , the  $f_i$  are IQL queries over  $ES_1$  and the  $g_i$  are IQL queries over  $I$ , then the pathway  $ES_1 \rightarrow ES_1 - I$  is automatically derived to be:

$$contract(c_1, Range Void Any), \dots, contract(c_m, Range Void Any)$$

### 2.3 Integration Workflow

Our techniques for creating federated, intersection and global schemas have been described in the previous subsection. These techniques now need to be incorporated into an overall workflow through which the global schema is produced iteratively. We describe the workflow below. Before doing so, we briefly describe the Intersection Schema Tool that has been developed to support this workflow. This tool firstly creates a single federated schema from a set of extensional schemas — this also serves as the first version of the global schema. The tool then allows a data integrator to incrementally identify semantic intersections between pairs of extensional schemas, and create a schema representing their intersection. Each intersection schema can be integrated with the current global schema using the tool, producing a new global schema.

The steps of the workflow proceed as follows:

1. Identify the extensional schemas representing the set of data sources that are to be integrated.
2. Initially a federated schema is created from the schemas identified in Step 1. Data Services can immediately be supported by this schema e.g. AutoMed's Query Tool [13].
3. Inspect the schemas identified in Step 1 and select two of them from which to derive an intersection schema.
4. Identify mappings between these two schemas and the intersection schema. For each Intersection Schema, a mappings table is maintained by the Intersection Schema Tool, which shows the IQL query correspondences between objects in the Intersection Schema and the current global schema. The Intersection Schema tool allows mappings to be added and edited by the data integrator. Other existing tools supported by AutoMed can be used to assist with this, for example the

Extent Tool which allows the extent of any schema object to be displayed [13] and the Schema Matching Tool [16] which aims to provide suggestions for schema mappings.

5. When all the mappings have been identified for this cycle of the workflow, the user can ask for an intersection schema to be generated. A new Global Schema is created automatically from the Intersection Schema and the extensional schemas by our tool, as described in Section 2.3. The user may optionally elect for any redundant objects in the new Global schema to be dropped.
6. The user may test the Intersection schema or Global schema at this stage by running queries on it using the AutoMed Query Tool and verifying that the results are as expected.

The data integrator can now proceed from Step 3 again, identify another pair of extensional schemas from which to create a new intersection schema, test this schema, generate a new global schema and, optionally, ask for any redundant objects to be removed from it. Any number of intersections between any pair of extensional schemas.  $ES_i$  and  $ES_j$  can be created at each iteration of the process.

## 2.4 Example

A large-scale data integration project that used AutoMed within a traditional data integration process was the iSpider project, which integrated data from several specialist Proteomics relational databases under a single virtual relational schema [19, 20]. In the iSpider project, all of the integration work was done “up front”, before any data services were deployed and the integration took several person months to accomplish.

Two of the source databases used in iSpider were Pedro<sup>2</sup> and PepSeeker<sup>3</sup> and we now illustrate how our intersection-schema based tools and methodology can be used to (partially) integrate them. We were able to do this by examining the set of schema transformations generated for the original iSpider project by the domain experts and data integrators working on that project. These are listed in Appendix E of [18].

An initial Federated schema is first generated, prior to the creation of any Intersection schemas. Upon inspection of the Pedro and Pepseeker fragments of this federated schema, the user identifies that  $\ll proteinhit, db\_search \gg$  from Pedro and  $\ll proteinhit, fileparameters \gg$  from PepSeeker are semantically equivalent concepts and should be represented by a new concept  $\ll UProteinHit, dbsearch \gg$  in an intersection schema between them<sup>4</sup>. The user creates an

<sup>2</sup><http://pedro.man.ac.uk/>

<sup>3</sup><http://nwsr.smith.man.ac.uk/pepseeker>

<sup>4</sup>AutoMed gives considerable flexibility for configuring how a construct  $m$  of a modelling language  $M$  is represented in the HDM, in general identifying the construct using a *scheme* of the form  $\ll M, m, s_1, \dots, s_n \gg$  where the  $s_i$  are either literals, or schemes representing other constructs. In this paper, we consider that AutoMed is configured to use a scheme of the form  $\ll sql, table, t \gg$  to represent an SQL ta-

intersection schema in the tool as illustrated in Figure 5. The left hand panel of the tool shows the source schemas for Pedro and PepSeeker and allows the user to select objects from each source schema. The bottom panel shows the transformation queries for each subset of objects selected, and the name of the new Global schema object. If only a single object is selected from a source schema then, by default, the tool automatically creates a transformation query consisting of just that object; the user is free to edit this query. The current Global schema is shown in the right hand panel. Once all the transformation queries in the “forwards” direction have been specified, a similar screen is presented to the user in order to specify the transformation queries in the reverse direction. This time, the top panel shows the newly defined Global schema objects on the left hand side and the source schemas on the right. Suggested transformation queries are presented in the bottom panel. Where possible, these queries are automatically generated by the tool from the user-specified queries in the forwards direction; for more complex transformations, the default query *Range Void Any* is used, which the user may edit. In this particular example, both the forwards and the reverse transformation queries consist of the same schema object. Once the user is satisfied with the new Global schema objects and the transformation queries, the user then requests the creation of the intersection schema.

The Intersection Schema is then automatically integrated with the original federated schema to create a new Global schema. If the user requests this, the schema objects  $\ll proteinhit, db\_search \gg$  from Pedro and  $\ll proteinhit, fileparameters \gg$  from PepSeeker can be dropped from the resulting federated schema, since their extents are included in the extent of  $\ll UProteinHit, dbsearch \gg$  in the Intersection Schema. Queries can now be run by the user over the resulting schema to verify the data integration, and in particular to check that the extent of  $\ll UProteinHit, dbsearch \gg$  is as expected.

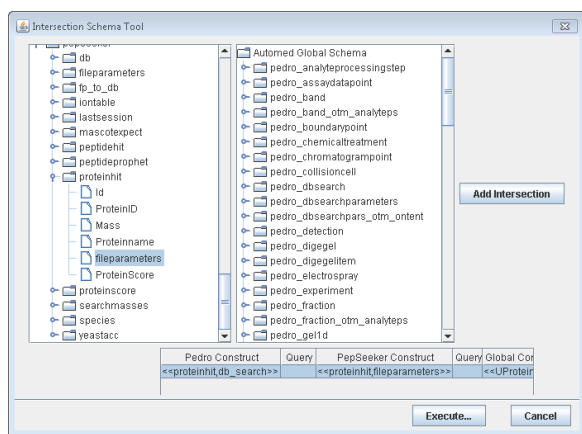
## 3. CASE STUDY

We have re-examined the iSpider documentation and the original set of schema transformations that were produced by that project in order to see how our intersection-schema based approach could have been used in the context of a large-scale real-world data integration project to undertake an incremental, pay-as-you-go integration that would have allowed query services to be supported in a gradual fashion. In addition to Pedro and PepSeeker, a third database that had been integrated with them in the iSpider project was gpmDB<sup>5</sup>.

The original iSpider project domain experts had identified a set of queries of high priority that the integrated resource should be able to answer (see Chapter 7 of [18]). Despite

able named  $t$ , and a scheme of the form  $\ll sql, column, t, c \gg$  to represent a column  $c$  of an SQL table  $t$ . Where the context of using a scheme would not cause ambiguity, the user may omit the modelling language  $M$  or the construct type  $m$  from the scheme, or both. Thus, in the context of the examples in this paper, where only relational tables are being considered, we refer to a table by a scheme of the form  $\ll t \gg$ , and to a column of a table by a scheme of the form  $\ll t, c \gg$ .

<sup>5</sup><http://www.thegpm.org/>



**Figure 5: Intersection Schema Tool - identifying mappings**

this fact, the iSpider project team elected to undertake a complete “up-front” integration of the data sources rather than using the set of queries to prioritise their integration effort.

We have used this set of queries in order to undertake an intersection schema-based integration, using the priority ordering of the queries to drive each iteration of the process. The set of queries are as follows, with higher priority queries being listed before lower priority ones:

1. Retrieve all protein identifications for a given protein accession number
2. Retrieve all protein identifications for a given group of proteins
3. Retrieve all protein identifications for a given organism
4. Retrieve all protein identifications given a certain peptide and their related amino acid information
5. Retrieve all identifications of a given protein given a certain peptide
6. Retrieve all peptide-related information for a given protein identification
7. Retrieve all ion related information

To answer query 1, we need to integrate  $\ll protein, accession\_num \gg$  from Pedro,  $\ll proseq, label \gg$  from gpmDB and  $\ll protein \gg$  from PepSeeker, to create an entity  $\ll UProtein, accession\_num \gg$  in the first intersection schema. These integrations are achieved by the following 6 transformations defined by the user in the Intersection Schema tool<sup>6</sup> (as well additional transformations automatically created by the tool<sup>7</sup>):

<sup>6</sup>The general syntax of the IQL queries within these *add* transformations is that of a *comprehension* [...] in which the expressions on the right hand side of the | are iterators and filters over one or more collections, and the expression on the left hand side of the | constructs a new collection.

<sup>7</sup>The additional *contract* transformations are automatically

```

Q1: {[an, lsid]}{[sid, an] ← ((UProteinHit, accession_number)); an = 'ENSP00000330074'}
Q2: {[an, lsid]}{[sid, an] ← ((UProteinHit, accession_number));
member [sid]{[sid, d] ← ((UProteinHit, description)); like d '%Actin%'} lsid}
Q3: {[an, lsid]}{[sid, an] ← ((UProteinHit, accession_number));
member [sid]{[sid, o] ← ((UProteinHit, organism)); like o '%sapiens%'} lsid}
Q4: {[pr, sc]}{[lsid1, pr] ← ((UProteinHit, protein));
[lsid2, seq] ← ((UPeptideHit, sequence)); seq = 'ATLTSDK';
{pepID, protID} ← ((UPeptideHitToProteinHit_mm));
lsid2 = pepID; lsid1 = protID;
[lsid2, sc] ← ((UPeptideHit, score));
{aid, pid} ← ((GS1_aa, GS1_pepid)); pid = pepID}
Q5: {[an, lsid1, sc]}{[lsid2, seq] ← ((UPeptideHit, sequence)); seq = 'LVNELTEFAK';
[lsid1, an] ← ((UProteinHit, accession_number)); an = 'gi-229552';
{pepID, protID} ← ((UPeptideHitToProteinHit_mm));
lsid2 = pepID; lsid1 = protID;
[lsid2, sc] ← ((UPeptideHit, score))}
Q6: {[an, seq, sc, pr, dbs]}{[lsid1, an] ← ((UProteinHit, accession_number));
lsid1 = '{URN:LSID:ispider.man.ac.uk:pedro'; 1069};
{pepID, protID} ← ((UPeptideHitToProteinHit_mm));
lsid1 = protID;
[lsid2, seq] ← ((UPeptideHit, sequence)); lsid2 = pepID;
[lsid2, sc] ← ((UPeptideHit, score));
[lsid2, pr] ← ((UPeptideHit, probability));
[lsid1, dbs] ← ((UProteinHit, dbsearch))}
Q7: {[ionid, mat, imm]}{ionid} ← ((GS1_iontable),
{ionid, mat} ← ((GS1_iontable, GS1_Matches),
{ionid, imm} ← ((GS1_iontable, GS1_Immon))}

```

**Table 1: Case Study Queries**

```

add << UProtein >> [{ 'PEDRO', k } | k ← << protein >>]
add << UProtein >> [{ 'gpmDB', k } |
k ← << proseq >>]
add << UProtein >> [{ 'pepSeeker', x } |
{k, x} ← << proteinhit, ProteinID >>]
add << UProtein, accession_num >> [{ 'PEDRO', k, x } |
{k, x} ← << protein, accession_num >>]
add << UProtein, accession_num >> [{ 'gpmDB', k, x } |
{k, x} ← << proseq, label >>]
add << UProtein, accession_num >> [{ 'pepSeeker', k, x } |
k ← << uprotein >>]

```

To answer query 2, we need additionally to create an attribute  $\ll UProtein, description \gg$  from  $\ll protein, description \gg$  in Pedro. This is achieved by the following transformation defined by the user<sup>8</sup>:

```

add << UProtein, description >> [{ 'PEDRO', k, x } |
{k, x} ← << protein, description >>]

```

To answer query 3, we need additionally to create an attribute  $\ll UProtein, organism \gg$  from  $\ll protein, organism \gg$  in Pedro. This is achieved by the following transformation defined by the user:

```

add << UProtein, organism >> [{ 'PEDRO', k, x } |
{k, x} ← << protein, organism >>]

```

To answer query 4, we additionally need to integrate the following:

generated. Where possible, the *delete* transformations in the pathway from a data source schema to an intersection schema are also automatically generated from the user-specified *add* transformations to complete the bidirectional pathway. For more complex transformations, the user’s input is needed to specify the *delete* transformations.

<sup>8</sup>Note, this is a refinement of the intersection schema-based methodology described in Section 2.3 — our Intersection Schema tool allows ad-hoc transformations of a single schema as well, as part of the iterative integration process.

- $\ll proteinhit, protein \gg$  from Pedro,  $\ll protein, proseqid \gg$  from gpmDB and  $\ll proteinhit, proteinid \gg$  from PepSeeker to create an attribute  $\ll UProteinHit, protein \gg$ ;
- $\ll peptidehit \gg$  from Pedro,  $\ll peptide \gg$  from gpmDB and  $\ll peptidehit \gg$  from PepSeeker to create an entity  $\ll UPeptideHit \gg$ ;
- $\ll peptidehit, sequence \gg$  from Pedro,  $\ll peptide, seq \gg$  from gpmDB and  $\ll peptidehit, pepseq \gg$  from PepSeeker to, create an attribute  $\ll UPeptideHit, sequence \gg$ ;
- $\ll peptidehit, score \gg$  from Pedro and  $\ll peptidehit, score \gg$  from PepSeeker to create attribute  $\ll UPeptideHit, score \gg$ ;
- $\ll proteinhit, db\_search \gg$  from Pedro and  $\ll proteinhit, fileparameters \gg$  from PepSeeker to create an attribute  $\ll UProteinHit, dbsearch \gg$ .

We also create an attribute  $\ll UPepTideHit, dbsearch \gg$  from  $\ll peptidehit, db\_search \gg$  in Pedro, and an entity  $\ll UPeptideHitToProteinHit\_mm \gg$  from the join of  $\ll UPeptideHit, dbsearch \gg$  and  $\ll UProteinHit, dbsearch \gg$  already in the global schema. This is achieved by the following 15 transformations defined by the user:

```

add  $\ll UProteinHit, protein \gg$  [ $\{ 'PEDRO', k, x \}$  |
   $\{ k, x \} \leftarrow \ll proteinhit, protein \gg$  ]
add  $\ll UProteinHit, protein \gg$  [ $\{ 'gpmDB', k, x \}$  |
   $\{ k, x \} \leftarrow \ll protein, proseqid \gg$  ]
add  $\ll UProteinHit, protein \gg$  [ $\{ 'pepSeeker', k, x \}$  |
   $\{ k, x \} \leftarrow \ll proteinhit, proteinid \gg$  ]
add  $\ll UPeptideHit \gg$  [ $\{ 'PEDRO', k \}$  |
   $k \leftarrow \ll peptidehit \gg$  ]
add  $\ll UPeptideHit \gg$  [ $\{ 'gpmDB', k \}$  |  $k \leftarrow \ll peptide \gg$  ]
add  $\ll UPeptideHit \gg$  [ $\{ 'pepSeeker', k \}$  |
   $k \leftarrow \ll peptidehit \gg$  ]
add  $\ll UPeptideHit, sequence \gg$  [ $\{ 'PEDRO', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, sequence \gg$  ]
add  $\ll UPeptideHit, sequence \gg$  [ $\{ 'gpmDB', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptide, seq \gg$  ]
add  $\ll UPeptideHit, sequence \gg$  [ $\{ 'pepSeeker', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, pepseq \gg$  ]
add  $\ll UPeptideHit, score \gg$  [ $\{ 'PEDRO', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, score >$  ]
add  $\ll UPeptideHit, score \gg$  [ $\{ 'pepSeeker', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, score \gg$  ]
add  $\ll UProteinHit, dbsearch \gg$  [ $\{ 'PEDRO', k, x \}$  |
   $\{ k, x \} \leftarrow \ll proteinhit, db\_search \gg$  ]
add  $\ll UProteinHit, dbsearch \gg$  [ $\{ 'pepSeeker', k, x \}$  |
   $\{ k, x \} \leftarrow \ll proteinhit, fileparameters \gg$  ]
add  $\ll UPeptideHit, dbsearch \gg$  [ $\{ 'PEDRO', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, db\_search \gg$  ]
add  $\ll uPeptideHitToProteinHit\_m \gg$  [ $\{ k1, k2 \}$  |
   $\{ k1, x \} \leftarrow \ll upeptidehit, dbsearch \gg$ ;
   $\{ k2, y \} \leftarrow \ll uproteinhit, dbsearch \gg$ ;  $x = y$  ]

```

To answer query 5, no further concepts need to be integrated. To answer query 6, we need to integrate  $\ll peptidehit, probability \gg$  from Pedro,  $\ll peptide, expect \gg$  from gpmDB and  $\ll peptidehit, expect \gg$  from

PepSeeker to create the attribute  $\ll UPeptideHit, probability \gg$  in the next intersection schema. This is achieved by the following 3 transformations defined by the user:

```

add  $\ll UPeptideHit, probability \gg$  [ $\{ 'PEDRO', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, probability \gg$  ]
add  $\ll UPeptideHit, probability \gg$  [ $\{ 'gpmDB', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptide, expect \gg$  ]
add  $\ll UPeptideHit, probability \gg$  [ $\{ 'pepSeeker', k, x \}$  |
   $\{ k, x \} \leftarrow \ll peptidehit, expect \gg$  ]

```

Finally, to answer query 7, no further concepts need to be integrated. For completeness, we list in Table 1 examples of the 7 queries (expressed in IQL) formulated over the resulting global schema.

In summary, we see that a total of  $6+1+1+15+3 = 26$  manually defined transformations are required to be able to answer all seven of these high priority queries.

By way of comparison, in the original iSpider integration (see Chapter 7 and Appendix E of [18] for full details, including listings of all the transformations), three successive versions of the global schema were produced, GS1, GS2, GS3. GS1 was defined to be identical with the Pedro schema due to the rich content of this schema compared with the gpmDB and PepSeeker schemas. AutoMed transformation pathways were then defined from the three data sources to GS1. Since all GS1 schema constructs have a trivial identify derivation from Pedro, we can consider the integration effort to comprise the manually defined transformations from gpmDB and PepSeeker to GS1. Also, we ignore any transformations whose query part is just *Range Void Any*. There are 19 non-trivial transformations from gpmDB to GS1 and 35 non-trivial transformations from PepSeeker to GS1.

The next version of the global schema, GS2, improved on GS1 by adding concepts that are supported by the gpmDB data source but not by Pedro and that therefore were not present in GS1. This required an additional 41 non-trivial transformations from PepSeeker to GS2 (note that all the additional transformations from Pedro to GS2 would have query parts *Range Void Any*).

The final version of the global schema, GS3, improved on GS2 by adding concepts that are supported by the PepSeeker data source but not by Pedro or gpmDB and that therefore were not present in GS2. This required no more non-trivial transformations (all the additional transformations from Pedro and gpmDB to GS3 would have query parts *Range Void Any*). Hence there are a total of  $19+35+41=95$  non-trivial transformations required by the original iSpider integration effort.

Of course, using the number of manually defined transformations as a comparison metric is rather crude; and moreover, as stated earlier, the original iSpider integration did not attempt to undertake a query-driven integration. Therefore, we are planning for the near future a more detailed evaluation of our intersection schema-based data integration methodology compared with traditional ones.

## 4. CONCLUSIONS

In this paper we have introduced a data integration methodology based on the concept of *Intersection Schemas*, using the AutoMed data integration framework. We have demonstrated the technique on a real-world data integration scenario, adopting a query-driven approach, and have seen that the number of user-defined steps required to perform the integration is significantly reduced compared to the original data integration methodology used by the domain experts on that project.

This work has been carried out in the context of the AutoMed data integration framework, which supports bidirectional schema and data transformations but which, up till now, has been used only for “up-front” data integration. In this paper we have shown how the AutoMed toolkit can be used to underpin a new light-weight data integration technique within an incremental pay-as-you-go data integration process, and hence how AutoMed can be applied within a dataspace environment.

Our future work includes extending the methodology so that intersections can be created between any number of source schemas at each iteration of the process, rather than just two as at present. We will also undertake a more detailed evaluation of our intersection-schema based integration approach with traditional integration methodologies in the context of further real-world large-scale data integration settings. For these investigations, we will take in both cases a query-driven approach and we will assess the productivity benefits arising using our approach. Since our techniques and tools are intended to be used by both domain experts and data integration experts, we propose a user evaluation which will consider two groups of users. The first group will consist of people familiar with the application domain but with limited knowledge of data integration processes, while the second group will be familiar with data integration processes but have limited knowledge of the application domain. Each group will be split randomly into two equally-sized subgroups. Each subgroup of a given group will be asked to undertake the same, query-driven, integration of a given set of data sources, guiding one subgroup through an intersection schema-based methodology and the other subgroup through a more traditional methodology, such as the ‘ladder’ approach [5]. A set of metrics will be measured for each subgroup, for example the time taken to complete the integration and the number of key clicks required within the toolset.

## 5. REFERENCES

- [1] B. Alexe, B. T. Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *ACM Trans. on Database Systems*, 36(4):23, 2011.
- [2] B. Alexe, L. Chiticariu, R. J. Miller, and W.-C. Tan. Muse: Mapping understanding and design by example. In *Proc. ICDE*, pages 10–19. IEEE, 2008.
- [3] B. Alexe, W.-C. Tan, and Y. Velegrakis. STBenchmark: towards a benchmark for mapping systems. *PVLDB*, 1(1):230–244, 2008.
- [4] B. Alexe, B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Designing and refining schema mappings via data examples. In *Proc. ACM SIGMOD*, pages 133–144. ACM, 2011.
- [5] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
- [6] M. Boyd, C. Lazanitis, S. Kittivoraviktul, P. Mc Brien, and N. Rizopoulos. An overview of the AutoMed Repository. *Technical Report, AutoMed Project*, 2004.
- [7] M. Franklin, A. Halevy, and D. Maier. From databases to dataspace: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.
- [8] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: the teenage years. In *Proc. VLDB*, pages 9–16. VLDB Endowment, 2006.
- [9] C. Hedeler, K. Belhajjame, N. W. Paton, A. Campi, A. A. Fernandes, and S. M. Embury. Flexible dataspace management through model management. In *Proc. EDBT/ICDT Workshops*, pages 114–134. Springer, 2010.
- [10] E. Jasper, A. Poulouvasilis, L. Zamboulis, and H. Fan. Processing IQL queries and migrating data in the automated toolkit. *Technical Report, AutoMed Project*, 2003.
- [11] E. Jasper, N. Tong, P. McBrien, and A. Poulouvasilis. Generating and optimising views from both as view data integration rules. In *Proc. DBIS’04*, volume 972, pages 13–30, 2004.
- [12] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. ACM PODS*, pages 233–246. ACM, 2002.
- [13] P. McBrien. AutoMed in a nutshell. *Technical Report, AutoMed Project*, 2006.
- [14] P. McBrien and A. Poulouvasilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE*, pages 227–238. IEEE, 2003.
- [15] P. McBrien and A. Poulouvasilis. P2P query reformulation over both-as-view data transformation rules. In *Proc. DBISP2P*, pages 310–322. Springer, 2006.
- [16] N. Rizopoulos. Automatic discovery of semantic relationships between schema elements. In *Proc. ICEIS (1)*, pages 3–8, 2004.
- [17] B. ten Cate, P. G. Kolaitis, and W.-C. Tan. Schema mappings and data examples. In *Proc. EDBT*, pages 777–780. ACM, 2013.
- [18] J. Wang. A Framework and Architecture for Quality Assessment in Data Integration. <http://www.dcs.bbk.ac.uk/research/recentphds/jwang.pdf>, 2012. [Online; accessed 01-December-2013].
- [19] L. Zamboulis, H. Fan, K. Belhajjame, J. Siepen, A. Jones, N. Martin, A. Poulouvasilis, S. Hubbard, S. M. Embury, and N. W. Paton. Data access and integration in the ISPIDER Proteomics Grid. In *Proc. Data Integration in the Life Sciences*, pages 3–18. Springer, 2006.
- [20] L. Zamboulis, N. Martin, and A. Poulouvasilis. Query performance evaluation of an architecture for fine-grained integration of heterogeneous grid data sources. *Future Generation Computer Systems*, 26(8):1073–1091, 2010.



# Bidirectional Transformations in Database Evolution: A Case Study "At Scale"

Mathieu Beine  
University of Namur  
Namur, Belgium  
math.beine@gmail.com

Nicolas Hames  
University of Namur  
Namur, Belgium  
nicolas.hames@gmail.com

Jens H. Weber  
University of Victoria  
Victoria, Canada  
jens@acm.org

Anthony Cleve  
University of Namur  
Namur, Belgium  
anthony.cleve@unamur.be

## ABSTRACT

Bidirectional transformations (BX) play an important role in database schema/application co-evolution. In earlier work, Terwilliger introduced the theoretical concept of a *Channel* as a BX-based mechanism to de-couple “virtual databases” used by the application code from the actual representation of the data maintained within the DBMS. In this paper, we report on considerations and experiences implementing such Channels in practice in the context of a complex real-world application, and with generative tool support. We focus on Channels implementing Pivot/Unpivot transformations. We present different alternatives for generating such Channels and discuss their performance characteristics at scale. We also present a transformational tool to generate these Channels.

## Keywords

Bidirectional transformations, database evolution, schema-code co-evolution, performance

## 1. INTRODUCTION

Many of today’s software applications are backed by database management systems (DBMS), most of them using a relational data model. With increasing system complexity and changing requirements arises the need to adapt and evolve software applications to meet new objectives. In the context of database applications, adaptations may be performed at the database level (i.e., schema changes, data migration) or at the level of the software application (i.e., program code). Changes made at either level often necessitate changes at the other level in order for the overall system to keep functioning. The synchronization of adaptation at different levels is often referred to as the schema/program co-evolution challenge.

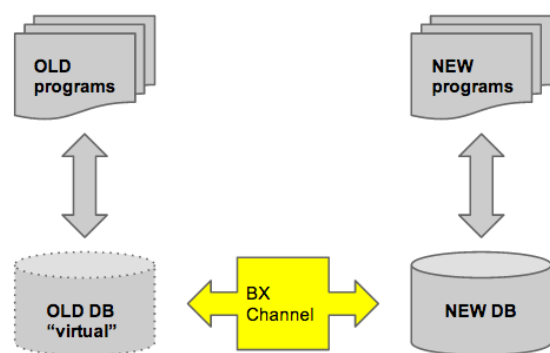


Figure 1: BX in DB/program co-evolution

Bidirectional transformations (BX) can be used as one way of addressing this co-evolution challenge. BX can be used to decouple the evolution of the database schema from the evolution of the program code, for example, by allowing changes to the database structure to be implemented while some programs can remain unchanged. In this case, any database access of the program code that uses the “old” schema is transformed to an equivalent database access using the new schema structure. Terwilliger [12] introduced the theoretical concept of a *Channel* to formalize this notion of transformations that translate application code queries to a “virtual database” structure to equivalent queries into the actual database implementation structure, cf. Figure 1.

From an engineering perspective, implementation of Channels in practice and at scale raises a range of design decisions and trade-offs. While previous authors have reported on experiences with implementing Channels (also referred to as wrappers) [3], such reports remain rare and often consider only small-scale applications and simple transformations only.

This paper presents empirical results from a large-scale industrial case study of engineering Channels to support the evolution of a complex medical information system. We focus on a couple of complex transformations, including *Pivot* and *Unpivot* and discuss their efficient implementation in practice. The Pivot and Unpivot operations can be described as rotation a table from a *1-column-per-attribute* to a

*1-row-per-attribute* representation and vice-versa. The reason why these operations are of particular interest from a software evolution point of view is that it is often beneficial to transform complex, sparsely populated table structures to a generic, more concise Entity-Attribute-Value (EAV) model. This reduces the complexity of the database schema as well as the programs accessing such data. The following case study will provide an example for such a transformation.

## 1.1 Case Study: OSCAR EMR Software

The application case study used in our work is a real-world medical information system called *OSCAR* used in primary health care in hundreds of clinics in Canada [8]. *OSCAR* has evolved over more than a decade and its current database includes more than 450 tables. The *OSCAR* database consists of well-populated “core” table structures that store information about patient demographics, allergies, medications, active problems etc., as well as more specialized, “satellite” table structures that store information for specific types of encounters and patient situations in primary care. These more specialized table structures are often associated with elaborate electronic *forms* that are filled out by the clinician on certain types of patient encounters. Due to the broad spectrum of different conditions that patients may have, these tables may have thousands of columns but any given data record (the actual data in each row) may only be populated sparsely (i.e., many null values).

The data in these large, sparsely populated tables are more adequately represented in a *1-row-per attribute* format, also called Entit-Attribute-Value (EAV) format, in order to save space and simplify program access. To see how the EAV representation might simplify program access, consider a program that exports all encounter information, including all forms that may exist for a given patient. Using the EAV model, such a program will not need to query a large set of different tables and probe the existence of values in each column.

*OSCAR*’s schema includes more than 60 form tables that can be transformed to a generic model in this way. This situation is by no means unique to our case study systems. Other systems and vendor products we have been working with show a similar structure and are expected to benefit from similar schema transformations.

## 1.2 Contributions and overview

This paper makes two main contributions. Firstly, we discuss implementation decisions and trade-offs related to the implementation of pivoting and unpivoting Channels at scale in the context of a real-world, industrial application. Secondly, we present a transformational tool for generating such Channel implementations in support of database schema evolution.

The rest of this paper is structured as follows. The following section provides an overview over research work related to our topic. Section 3 defines the transformations used in our work in more detail. Section 4 discusses implementation alternatives and trade-offs realizing the pivoting Channel. Section 5 presents the transformational tool we implemented for generating Channel implementations. Section 6 presents quantitative results from studying the performance of Channel implementations. Finally, Section 7 offers conclusions and directions for current and future work.

## 2. RELATED WORK

Most software must continue to adapt to fit changing requirements to remain useful. Database applications are no exception. In the context of database evolution, we are primarily interested in changes that involve the database schema definition. Evolution of program code that does not impact the database is out of scope for this paper and subject to a broad spectrum of research on different aspects of software evolution and reengineering. The reader may refer to [10] for a general overview.

Changes to the schema definition of a database application usually (but not always) require updates to application code (programs) that use the database as well as updates to the actual data instances. These two kinds of updates are commonly referred to as *application migration* and *data migration*, respectively [9]. A common strategy for adapting application programs to database changes is to use so-called *wrappers*, i.e., programs that “hide” the database changes from the application program by effectively translating queries (and updates) of the “old” database to equivalent accesses to the “new” (changed) database [13]. Application programs adapted in such a way can remain unchanged. Conversely, wrappers can also be used to accommodate evolution in application programs, while leaving the database implementation unchanged. In that case, the wrapper will transform queries (and updates) from newly developed or evolved programs (requiring a modified database structure) to accesses on the “old” database implementation.

In practice, large-scale database applications that have evolved over longer periods of time often have to support a combination of wrappers for forward as well as backward compatibility of different versions of programs with different versions of databases.

Of course, the question as to whether or not it is possible to create a wrapper that adapts a particular program (version) to a particular version of a schema depends on the nature of changes made in the schema (or the program). Schema changes are usually formalized in terms of transformation functions and categorized as information capacity preserving, -augmenting, -reducing. Thiran et al. [13] have proposed a semi-automatic approach for generating wrappers from composition of well-defined schema transformations. They report experiences with wrapping a small and a medium size system, but do not consider performance characteristics or more complex transformations, such as the ones discussed in this paper.

In later work, Terwilliger extends the concept of database wrappers to that of so-called *Channels* [12]. The latter not only transforms data queries and manipulations (queries / inserts / updates) from a “virtual” database to the real database, but also transforms schema manipulations in a similar manner. In other words, from the point of view of an application program, a Channel should be indistinguishable from a “real” database. While Terwilliger discusses many of the same transformations as Thiran et al. (e.g., table partitioning and merging), he also discusses *Pivot* and *Unpivot* transformations. These two transformations are of particular importance for database evolution and studies of their efficient implementation in auto-generated wrappers (or channels) with “at-scale” systems are scarce. We therefore concentrate on these transformations in this paper.

Research on BX has not been confined to the domain of databases, but other communities such as software engi-



neering, programming languages and graph transformations have studied BX based on different theoretical frameworks. Czarnecki et al. [4] provide an overview and comparison of BX theories across disciplines. A theoretical framework of particular influence on the BX community has emerged from programming language domain, namely the lens framework [6]. In its most basic form, BX are considered as pairs of functions commonly referred to as  $get : S \rightarrow V$  and  $put : S \rightarrow V \rightarrow S$ , where the first one produces a view  $V$  on a source data structure  $S$  and the second one updates the source data structure  $S$  according to any changes made to  $V$ . The special case of applying  $put$  to an empty source model in  $S$  is also referred to as  $create$ , i.e.,  $create(v) \equiv put(v, \emptyset)$ . While our work on Channels is not formally based on the theory of lenses, we will informally adopt the  $get/put/create$  terminology framework [11] to describe the transformation that creates a virtual database ( $get$ ) for the purpose of legacy program access and propagates any updates to that virtual DB to the real database ( $put$ ) (cf. Figure 1).

Research on generating bidirectional channels (or wrappers) is related to the well-known *view-update problem* in databases [1]. Bohannon et al. [2] present *relational lenses* as an attempt to adapt the lens framework to address the relational view update problem from an algebraic perspective. They propose a new programming language to formalize view update policies in terms of lenses and define formal laws on well-behavedness of these lenses. While related, our work on database schema evolution has a different objective. Rather than programming BX, we are interested in automatically generating BX Channels as a side effect of applying schema redesign transformations during the process of evolving and refactoring database applications.

### 3. TRANSFORMATION DEFINITION

In this section, we provide definitions for the primitive and composite transformation used in this paper.

#### 3.1 Pivot and Unpivot

The *Pivot* operator ( $T' = \text{PIVOT}(T, A, V)$ ) transforms a table  $T$  in generic key-attribute-value form into a form with one column per attribute. Column  $A$  must participate in the primary key of  $T$  and provide the names for the new columns in  $T'$ , populated with data from column  $V$ . The resulting table is named  $T'$ . The formal definition given for the Pivot operator using relational algebra is presented below and based on [12]. Readers who are unfamiliar with relational algebra are referred to [7] for a primer.

$$\begin{aligned} \not\bowtie_{C;A;V} T &\equiv \\ (\pi_{columns(T) - \{A, V\}} T) \bowtie (\rho_{V \rightarrow C_1} \pi_{columns(T) - \{A\}} \sigma_{A=C_1} T) \\ \bowtie \dots \bowtie (\rho_{V \rightarrow C_n} \pi_{columns(T) - \{A\}} \sigma_{A=C_n} T) \\ \text{for } C_1, \dots, C_n = C = \delta(\pi_A(T)) \end{aligned}$$

Figure 2 shows the intermediate steps of the Pivot operation for an example table  $T$  with three columns. In this case, *Period* is the pivoting attribute  $A$  whose values will give rise to columns in the resulting table and *Price* provides the values for these columns. Figure 2 shows that intermediate tables are created for each arising attribute. The key for the resulting table  $T'$  will be all remaining columns in  $T$  (all columns other than  $A$  and  $V$ ).

The *Unpivot* operator ( $T' = \text{UNPIVOT}(T, A, V)$ ) is the in-

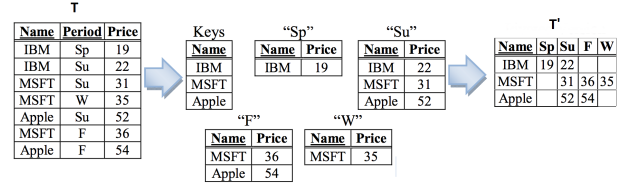


Figure 2:  $T' = \text{PIVOT}(T, \text{Period}, \text{Price})$

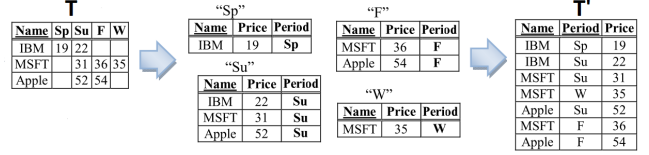


Figure 3:  $T' = \text{UNPIVOT}(T, \text{Period}, \text{Price})$

verse of the Pivot operator and transforms a table  $T$  from a one-column-per-attribute form into key-attribute-value triples, effectively moving column names into data values in new column  $A$  (which is added to the key) with corresponding data values placed in column  $V$ . The resulting table is named  $T'$ . The formal definition (given in [12]) for this operator in relational algebra is presented below.

$$\begin{aligned} \not\bowtie_{C;A;V} T &\equiv \\ \bigcup_{C \in C} (\rho_{C \rightarrow V} \pi_{columns(T) - (C - \{C\})} \sigma_{C <> null}(T)) \\ &\times \rho_{1 \rightarrow A}(name(C)) \end{aligned}$$

Figure 3 shows the intermediate steps of the Unpivot operation.

#### 3.2 VPartition and VMerge

In practice, Pivot and Unpivot transformations are often used in composition with two other operators, commonly referred to as *VPartition* and *VMerge* in [12]. The  $(T_1, T_2) = VPartition(T, f)$  operator splits a given table into two tables  $T_1, T_2$ , according to a total selection function  $f$ , which associates each non-key column with one of the two target tables ( $T_1$  or  $T_2$ ). Both resulting tables share the key columns of  $T$ . The  $(T' = VMerge(T_1, T_2))$  operator is the inverse of the *VPartition* operator and reconstructs a single table using two tables sharing a common primary key. The formal definition of *VPartition* and *VMerge* is straight forward based on projection and joins in the relational algebra, respectively, and omitted here.

#### 3.3 Complex transformations: create/get/put

Complex transformations (and the Channels that implement them) can be composed by concatenations of primitive ones, such as the ones defined above. The composite transformation we will focus on in our case study combines *VPartition* and *Unpivot* to transform database structures into one-column-per-attribute format into equivalent structures into an Entity-Attribute-Value (one-row-per-attribute) format (akin to *create* and *put* in the lens framework). The inverse transformation composes Pivot and *VMerge* to re-

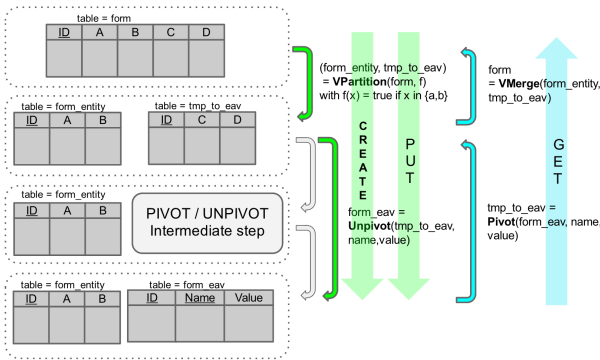


Figure 4: Composite BX - create/get/put

construct the original structure (akin to *get*). Figure 4 illustrates these transformations with a graphical example.

## 4. CHANNEL IMPLEMENTATION

Different strategies and techniques can be applied when implementing Channels for the above transformations. This section describes and compares several such alternatives and presents a novel technique referred to as the “coalescing approach”. For this discussion, we assume that the database management system (DBMS) used does not have built-in operators for Pivot and Unpivot transformations. Indeed, most current DBMS still lack these operators.

In most of the examples below, we present some SQL pseudo-code in order to help the reader to fully understand the theory. All the examples are based on the tables from the schema available in the Figure 4.

### 4.1 Implementing "Create"

In our application domain of database evolution, *create* is mainly used for the data migration task, i.e., to transform data that conform to the “old” schema to equivalent data conforming to the newly evolved (transformed) schema. The amount of that data may be large in real-life applications.

This step can be implemented with a DB client program or directly within the DB server, in the form of a stored procedure. We implemented both alternatives. As expected, the first alternative is much less efficient. However, it has the benefit of being more platform independent. The algorithms are similar for both approaches and provided below.

#### 4.1.1 The procedural approach

The procedural approach uses nested loops. The first loop inserts each entity in the entity table (the table containing the columns that will not be unpivoted, e.g., the entity keys and any columns that should remain in the original format). For each inserted entity, the second loop will be executed in order to insert the unpivoted attributes in the corresponding Entity Attribute Value (EAV) table. An example using the tables from Figure 4 is presented at Figure 5. Although it is a small example, it is easy for the reader to project this for such big tables as exist in real software systems.

#### 4.1.2 The declarative approach

The declarative approach first inserts all the entities into the entity table and then it executes one “big” insert composed of unions of select to migrate all the unpivoted at-

```

BEGIN
DECLARE id_var,A_var,B_var,C_var,D_var INTEGER;
DECLARE cur1 CURSOR FOR SELECT * FROM form;
OPEN cur1;
  read_loop: LOOP
  FETCH cur1 INTO id_var,A_var,B_var,C_var,D_var;
  IF (no more records){LEAVE read_loop;}
  INSERT INTO form_entity VALUES(id_var,A_var,B_var);
  ...
  IF(C_var is not null or C_var !="){
  INSERT INTO form_eav VALUES(id_var,"C",C_var);}
  ... (for all the unpivoted attributes)
  END LOOP;
CLOSE cur1;
END

```

Figure 5: Procedural approach (Create)

```

BEGIN
INSERT INTO form_entity SELECT id,A,B FROM form;
INSERT INTO form_eav SELECT * from (
SELECT id,name,value FROM (SELECT id,C AS value FROM
form WHERE C IS NOT NULL),(SELECT "C" AS name
FROM DUAL)
UNION
SELECT id,name,value FROM (SELECT id,D AS value FROM
form WHERE D IS NOT NULL),(SELECT "D" AS name
FROM DUAL))
END

```

Figure 6: Declarative approach (Create)

tributes into the corresponding table. The SQL pseudocode is given at Figure 6.

## 4.2 Implementing "Put"

The solution presented here is an implementation of the update channel transformation defined by Terwilliger [12]. The update channel transformation consists of three basic operations that are insert, update and delete. Database *triggers* are a natural solution for invoking these operations. Each time a “legacy” application inserts/updates/deletes the data in the virtualized (old) DB, a dedicated trigger executes the corresponding part of the *put* function. The listings at Figures 7, 8 and 9 sketch the SQL pseudocode for the insert, delete and update triggers in our example.

Of course, these triggered *put* functions may fail if some integrity constraints become violated due to concurrent updates, e.g., an insert fails if an item with the same key already exists in the target database. We do not further discuss concurrency issues in this paper, as it is assumed that legacy programs use transactions when accessing the virtualized database, and executing the Channel code is part of that

```

% SQL Code for Insert trigger
CREATE TRIGGER insert_form INSTEAD OF INSERT ON
form_view
FOR EACH ROW BEGIN
INSERT INTO form_entity VALUES(NEW.id,NEW.A,NEW.B);
...
IF(NEW.C IS NOT NULL){
INSERT INTO form_eav VALUES(NEW.id,"C",NEW.C);}
... (for all the unpivoted attributes)
END

```

Figure 7: Insert trigger (Put)

```

% SQL Code for Delete trigger
DROP TRIGGER IF EXISTS delete_form;
CREATE TRIGGER delete_form INSTEAD OF DELETE ON
  form_view
FOR EACH ROW
BEGIN
  DELETE FROM form_eav WHERE id=OLD.id;
  DELETE FROM form_entity WHERE id=OLD.id;
END

```

Figure 8: Delete trigger (Put)

```

% SQL Code for Update trigger
CREATE TRIGGER update_form INSTEAD OF UPDATE ON
  form_view
FOR EACH ROW BEGIN
UPDATE form_entity SET B=NEW.B,A=NEW.A where id=NEW.
  id;
...
IF (NEW.C not like OLD.C){
  IF (OLD.C!=""){
    INSERT INTO form_eav VALUES(NEW.ID,"C",NEW.C);
  }ELSEIF(new.id is null){
    DELETE FROM form_eav WHERE name="C" AND ID=NEW.ID;
  }ELSE
    UPDATE form_eav SET value=NEW.C WHERE name="C" AND
      ID=NEW.id;
  }
}
... (for all the unpivoted attributes)
END

```

Figure 9: Update trigger (Put)

transaction (and potentially aborts it in case of conflicts). Pessimistic strategies (locking) may be used to avoid such inconsistencies at the cost of limiting concurrency. However, locks applied on the virtualized DB should be propagated through the Channel to the actual DB to be effective. Terwilliger’s current model of Channels does not cover the propagation of locks, nor does our implementation of Channel transformations [12]. We will address this limitation in future work.

### 4.3 Implementing "Get"

The *get* function recreates the old “virtual” database for the legacy programs to use. To implement the *get* function, we first followed the formal definition presented in Section 3.1. However, we found scalability problems with this solution. We therefore present a second implementation right after to avoid these problems.

#### 4.3.1 The join approach

In order to allow “legacy programs” to keep working on the virtualized “old” database, we defined a set of queries that can be used to define views on the database. We implemented DML-SQL triggers to handle the usual CRUD operation on the new schema through the “virtual schema”. This implementation is based on the theoretical solution described in [12].

- *The Join approach* The join approach creates for each column that was unpivoted from the original table, an intermediate table containing the key-attribute-value triple for all the non-null values in the original table. Those intermediate tables are then joined together in order to create our “virtual schema”. A pseudocode

```

SELECT a.id,a.A,a.B,a1.value as C,a2.value as D
FROM form_entity a
LEFT OUTER JOIN form_eav as a1
  ON a1.id=a.id
  AND a1.name="C"
LEFT OUTER JOIN form_eav as a2
  ON a2.id=a.id
  AND a2.name="D";

```

Figure 10: Join approach (Get)

example is given at Figure 10.

This approach is theoretically perfect. However, at scale, we found that DBMS run into the problem of the maximum-joins-per-query limit. The DBMS used in our case study application (MySQL) allows 61 joins per query. Some DBMS have higher limits, such as Microsoft’s SQLServer, which accepts up to 256 joins. However, some of the tables in our case study have thousands of columns, which would require thousands of joins, clearly exceeding such a limit.

- *The join approach revisited* A solution to face the maximum-joins-per-query limit is to split the set of columns to migrate into multiple subsets, execute the join approach for subsets having less than 61 columns (or whatever the join limit of the DBMS may be) and then joining all those subsets in the final table. This solution worked at scale but lacked in performance compared to the *coalescing* approach we will describe below.

#### 4.3.2 The coalescing approach

We decided to design another solution to execute the Pivot operation. We refer to this solution as the “coalescing approach”. The formal definition of the Unpivot operator for the coalescing approach is given below:

$$\begin{aligned}
\tilde{\rho}_{\mathbf{C};A;V}(T) &= \gamma_{(columns(T)-\{A,V\}),MAX(C_1),\dots,MAX(C_n)} \\
&((\bigcup_{c \in \mathbf{C}} (\rho_{value \rightarrow name(C)}(\pi_{value,id}(\sigma_{A=name(C)}(T)))))) \\
&\times (\rho_{1 \rightarrow name(C'_1)}(null) \bowtie \dots \bowtie \rho_{1 \rightarrow name(C'_n)}(null)) \\
&\text{for } C_1, \dots, C_n = \mathbf{C} \text{ AND } C'_1, \dots, C'_n = \mathbf{C} - \{C\}
\end{aligned}$$

where  $\mathbf{C}$  is the set of values on which to pivot (the set of attributes you want to pivot),  $A$  is the Pivot column (the column containing the values for the new column names) and  $V$  is the pivot-value column (the column containing the value for the attributes).

This operation can be decomposed in multiple intermediate steps that will be explained here.

For this operation, the query is executed with a group-by clause on the id on the EAV table.

First, the query selects each row in the EAV table (relation  $T$ ) where the column  $A$  contains the name of a column that belongs to  $\mathbf{C}$  (In the example given below,  $C = \{(columns(T)-\{A,V\})\}$ , i.e. all columns are pivoted.) and transform it from a 1-row-per-attribute to a 1-column-per-attribute representation. This will create as many rows as they are attributes for the given id in the EAV model.

```

SELECT id,A,B,C,D FROM
(SELECT
MAX(IF(a.name ='C',a.value,null)) AS 'C',
MAX(IF(a.name ='D',a.value,null)) AS 'D',
id FROM form_eav a GROUP BY id) AS grp, form_entity
AS i WHERE i.id=grp.id;

```

Figure 11: Coalescing approach (Get)

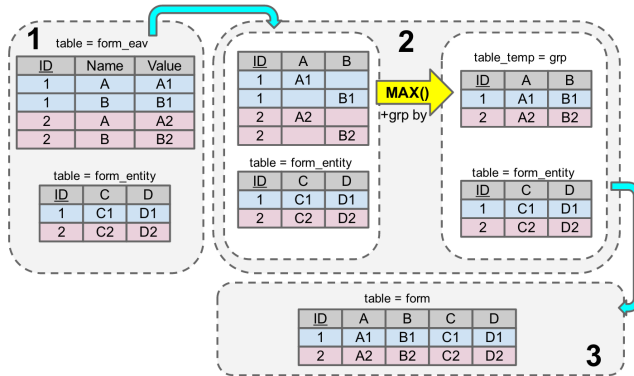


Figure 12: The coalescing approach

Then, for each row, the query will add columns that exists in the destination schema (all the values of C-A) and put a “null” value into those “joined” columns. This will produce a row having only one attribute with a non-null value per row and null-values for all the other columns.

Finally, the query executes an aggregate function (MAX) in order to “coalesce” all the rows corresponding to same id into only one row.

This approach is significantly faster than the join approaches. We only have to execute one select for each entity and then execute only one join with the entity table, as shown at Figure 11. Figure 12 illustrates the coalescing algorithm described above. The example starts with the EAV model and reconstructs the original table, i.e., the virtual database. On the left (box 1), there is the EAV table and the table containing some columns kept in a 1-column-per-attribute representation. In the middle of the figure, box 2 presents the operation of pivoting the EAV table and joining the result with the entity table. The result of this box is the original table, or virtual database, presented in the box 3.

The aim of this MAX function is to coalesce all the rows that contains only one non-null value per row for each id into only one row per id, and so allow us to retrieve the original table. The example of Figure 13 (subschema of Figure 12) depicts the application of the max operator in this specific case. Here, the max function is used to retrieve the only “non-null” value for a specific column of a given id value.

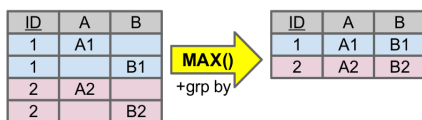


Figure 13: The MAX function

#### 4.4 “Type-preservation” EAV model

One issue arising with transforming relational data into an EAV model (unpivoting) and back (pivoting) is the preservation of type information. Columns in the original table may use a large variety of different types. However, once cast into a joint EAV model, that type information may be lost, if it is not preserved. The implementation of type-preservation may complicate the resulting EAV model. We can consider different implementation alternatives summarized below. The three first alternatives have been described in [5].

1. The basic EAV schema. This schema store all the values in a single columns that usually uses a generic TEXT data-type. The original type information is lost.
2. The multi data-type EAV Schema. This schema use multiple tables, one for each data-type.
3. The hybrid EAV schema. This schema use multiple columns in the EAV table, one column for each data-type.
4. The Variant data-type. This schema use a variant data-type to store the different data-type. This solution has performance limitations and may not be offered in many DBMS systems. (It is not offered in MySQL, for example, the DBMS used by OSCAR.)

The choice we made for the Oscar case is to use a hybrid EAV schema with on table and multiple columns types. Since this may result in a potentially large number of columns, our transformation implementation generates an EAV schema to consider only those datatype that are really needed in the original tables.

This issue of type-preservation also implied that it is impossible to use the PIVOT and UNPIVOT functions that are sometimes defined in certain DBMS. As soon as we have to manage multiple column in the input for the pivot function or in the output for the unpivot function, we have to define our own implementation.

Another detractor of using PIVOT/UNPIVOT operators provided by some DBMS is that they are not well-defined and lack a unified semantics (see [14]). It is therefore not possible to predict what will be the output in specific cases as for example, a non-unique id for the pivot function.

## 5. TOOL SUPPORT

We developed a plug-in for DBMain([www.db-main.eu](http://www.db-main.eu)), an interactive database (re)engineering tool developed by the University of Namur and its spin-off company Rever. To date, DBMain offers rich support for database schema transformations, but does not generate Channels. Our plug-in extends DBMain with the capability of evolving database schemas based on the aforementioned transformations. DBMain generates the database definition of the newly evolved schema as well as all the code for the bidirectional Channel that allows legacy programs to run on the newly evolved database. We have experimented with different alternatives to implement the required transformations, particularly Pivot and Unpivot, as these operators are not provided as built-in primitives by most database management systems, or at least, not as we had to use it.

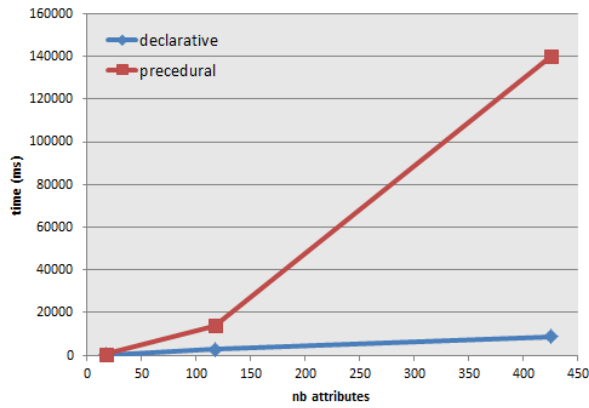


Figure 14: Create performance

The DBMain plugin can be used to pivot/unpivot tables from a DBMain schema (SQL to DBMain schema extraction also available in DBMain), one at a time or multiple at a time. The plugin then supports the migration of the existing data into the new EAV schema, generate the channel implementation (triggers), generate the view’s, test the data migration, etc.

The tool support provided by the DBMain plugin is significant for two reasons, *correctness* and *scalability*. The first reason is related to the safety critical nature of healthcare information systems. There are strict requirements on the correctness of transformations and the ability for backwards compatibility of programs that use the “old” database structures. A tool that is capable of implementing schema transformations based on formally defined lossless transformations as well as generating code for Channels that can be used to automatically adapt “legacy programs” provides welcome assurance in this domain.

Secondly, the size of the Channel code generated by our plugin is considerable and writing this code by hand would be tedious and error prone. We found that in the best performing code (discussed below), each column in a transformed table gives rise to approximately 34 lines of code in the update (“put”) function of the channel. A table of 1000 columns will therefore give rise to 34KLOC of channel code for the “put” direction alone.

## 6. IMPLEMENTATION COMPARISON

In order to evaluate the viability of our solution in a real world application we decided to perform some performance tests. We will provide here some performance measurements of the different implementations presented above.

First, we present a comparison of the performance for the data migration from the original model to the EAV format. This step corresponds to the implementation of the Channel *create* function. It is composed of a VPartition operation followed by the Unpivot operation. For these performance tests we decided to benchmark the data migration time on three different tables. We choose three tables from OSCAR containing 17, 117 and 425 columns. The following chart gives an overview of the time needed to migrate 1000 records from the original table to the EAV table.

Figure 14 shows the different performance characteristics of the declarative implementation and the procedural meth-

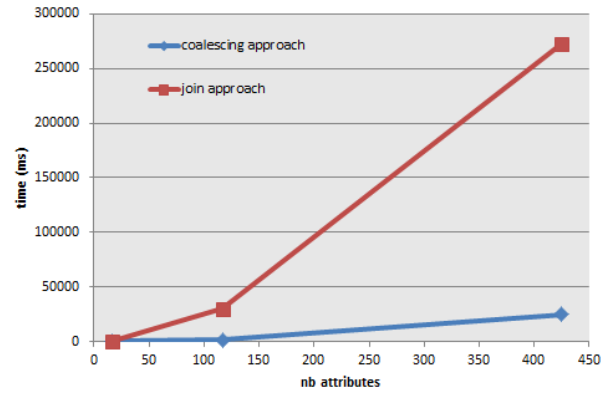


Figure 15: Pivot performance

ods.

The performance gap between the two unpivoting methods can be understood by taking a look on the SQL query. The procedural statement executes a query for each value of each line that have to be unpivoted to insert all the field value one by one. The DBMS query optimizer is not able to optimize this iterative loop. In the other hand, the declarative approach executes only one insert query. This query is composed of one sub-query per attribute, but is not directly dependent of the number of rows contained in the original table, even if it will impact the data set size. The DBMS query optimizer can optimize and execute this single nested query more efficiently.

We also took measurements on the view reconstruction query (“get”). First, a Pivot operation is executed and then a VMerge is applied on the result of the Pivot operation with the table that contains the attributes kept in the “classical” relational form. We present in the Figure 15 the time to pivot/merge the table for the join approach and the coalescing approach. For these measurements we took the same tables as above and measured the time needed to perform a select query on the EAV model containing 1000 entities.

Comparing the performances of the two pivoting methods, we see that the first method uses a lot of joins (costly database operation), by creating one temporary sub-table per pivoted attribute. The second approach (coalescing approach) performs a unique select query that retrieves a huge result-set, then manipulates it to pivot the data. This method performs no join nor any costly operator. It only uses a single select with some conditions and is therefore faster.

## 7. CONCLUSION

Database evolution raises the challenge of co-evolving all program code that uses the database, unless we can put in place “adapters” that allow programs to remain unchanged and use the database in its “old format”. Bidirectional transformations (BX) and Channels implementing BX can play an important role in keeping legacy applications running while evolving the database to a more suitable structure. In this paper, we have reported on experiences of generating Channels for an industrial case study “at scale”. In particular, we focus on Channels involving transformations between traditional relational structures (one column per attribute) and generic project data structures (one row per attribute).

Implementation alternatives of these transformations have not been studied at scale to date. We present performance and salability aspects related to different implementation techniques and propose a novel approach for implementing the Pivot operator, referred to as the coalescing technique. We developed a plug-in for DBMain that extends the database reengineering tool with capabilities of generating Channel implementation code. Our future work is on researching ways in which Channel transformations can be implemented by means of object-relational mapping descriptions. Current object-relational middleware does not have support for complex transformations, such as Pivot and Unpivot, and would have to be extended to implement such Channels.

## 8. REFERENCES

- [1] F. Bancilhon and N. Spyrtatos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, Dec. 1981.
- [2] A. Bohannon, B. C. Pierce, and J. A. Vaughan. Relational lenses: A language for updatable views. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, pages 338–347, New York, NY, USA, 2006. ACM.
- [3] A. Cleve, J. Henrard, D. Roland, and J.-L. Hainaut. Wrapper-based system evolution - application to CODASYL to relational migration. In K. Kontogiannis, C. Tjortjis, and A. Winter, editors, *Proceedings of the 12th European Conference in Software Maintenance and Reengineering (CSMR 2008)*, pages 13–22. IEEE Computer Society, 2008.
- [4] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Proceedings of the 2nd International Conference on Theory and Practice of Model Transformations*, ICMT '09, pages 260–283, Berlin, Heidelberg, 2009. Springer-Verlag.
- [5] S. El-Sappagh, S. El-Masri, A. M. Riad, and M. Elmogy. Electronic health record, data model optimized for knowledge discovery. *International Journal of Computer Science issues*, 9, 2012.
- [6] J. N. Foster, A. Pilkiewicz, and B. C. Pierce. Quotient lenses. *ACM Sigplan Notices*, 43(9):383–396, 2008.
- [7] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.
- [8] M. Gobert, J. Maes, A. Cleve, and J. Weber. Understanding schema evolution as a basis for database reengineering. In *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM 2013)*. IEEE Computer Society, 2013.
- [9] J.-L. Hainaut, A. Cleve, J. Henrard, and J.-M. Hick. Migration of legacy information systems. In *Software Evolution*, pages 105–138. Springer Berlin Heidelberg, Jan. 2008.
- [10] T. Mens and S. Demeyer, editors. *Software Evolution*. 2008.
- [11] J. Terwilliger, A. Cleve, and C. Curino. How clean is your sandbox? : Towards a unified theoretical framework for incremental bidirectional transformations. In *Proceedings of the 5th International Conference on Model Transformation (ICMT 2012)*, volume 7307 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2012.
- [12] J. F. Terwilliger. Bidirectional by necessity: Data persistence and adaptability for evolving application development. In *GTTSE*, pages 219–270, 2011.
- [13] P. Thiran, J.-L. Hainaut, G.-J. Houben, and D. Benslimane. Wrapper-based evolution of legacy information systems. *ACM Trans. Softw. Eng. Methodol.*, 15(4):329–359, Oct. 2006.
- [14] C. M. Wyss and E. L. Robertson. A formal characterization of pivot/unpivot. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 602–608, New York, NY, USA, 2005. ACM.



# Entangled State Monads

## Extended abstract

James Cheney, James McKinna,  
Perdita Stevens

School of Informatics, University of Edinburgh  
firstname.lastname@ed.ac.uk

Jeremy Gibbons, Faris Abou-Saleh

Dept. of Computer Science, University of Oxford  
firstname.lastname@cs.ox.ac.uk

### ABSTRACT

We present a monadic treatment of symmetric state-based bidirectional transformations, and show how it arises naturally from the well-known asymmetric lens-based account. We introduce two presentations of a concept we dub the “entangled” state monad, and prove their equivalence. As a step towards a unifying account of bidirectionality in general, we exhibit existing classes of state-based approaches from the literature as instances of our new constructions. This extended abstract reports on work in progress.

### 1. INTRODUCTION

This extended abstract describes work in progress towards unifying approaches to formalising bidirectional transformations (bx). For purposes of this paper, a bx is a device for maintaining consistency between two or more information sources. In model driven development, such sources are usually models; for example, UML models of a system to be developed. Other artefacts treated with these techniques could include database tables, XML files, abstract syntax trees, code, etc. We use the (admittedly overloaded) term ‘models’ broadly to refer to any of these information sources.

There are multiple dimensions over which notions of bx vary. For example, they may operate on only two information sources, or several. They may insist that one source be a strict abstraction of the others (asymmetric case), or not (symmetric case).

Our main motivation is to lay foundations that we will later use to work towards a uniform, typed understanding of the extra information that is used by bx, besides the current states of the models that are to be synchronised. We begin in this paper with state-based bx, including those with explicit complement.

In formal semantics, stateful computations are often expressed in terms of *monads* [3], giving a unified account of impure side-effects in pure functional languages. They have since become an essential programming pattern in such languages [6], and we follow suit.

### 2. BACKGROUND

*Monads for Effectful Functional Programming.* The essential idea of monads in functional programming is to encapsulate

a computation with side-effects, taking inputs of type  $A$  and returning a result of type  $B$ , as a function of type  $A \rightarrow MB$ , for a suitable type constructor  $M$ , known as a monad. Whereas inhabitants of the plain type  $A$  denote pure values, those of the monadic type  $MA$  denote *computations*, which may incur computational effects before yielding a value of type  $A$ . For instance, one may describe non-deterministic computations of type  $A \rightarrow B$  in terms of the *List* monad – i.e., as functions  $A \rightarrow List B$ , where each value  $a : A$  is assigned a list of possible return values  $[b_1, b_2, \dots] : List B$ . Monads can be used to capture side-effects, input/output, exceptions, probabilistic choice, and many other computational effects. In this paper we are concerned with computations which may depend on, and modify, various forms of mutable state; such computations are described by the *state* monad, as defined shortly.

More formally, a *monad* is a type constructor  $M$  equipped with the following structure of typed operations (parametric in  $A, B$ ):

$$\begin{aligned} \text{return} & : A \rightarrow MA \\ (\gg\equiv) & : MA \rightarrow (A \rightarrow MB) \rightarrow MB \\ (\gg) & : MA \rightarrow MB \rightarrow MB \\ ma \gg mb & = ma \gg\equiv \lambda\_ . mb \end{aligned}$$

(We borrow the Haskell convention of writing an infix operator  $\oplus$  in parentheses ( $\oplus$ ) in order to refer to it without arguments.) Here, the operation *return* simply returns its argument with no other effect. The ‘bind’ operation  $ma \gg\equiv f$  runs a computation  $ma$  returning an  $A$ , then runs a computation  $f$ , parameterized over  $A$  and returning a  $B$ , finally returning that  $B$  value. The definable operation ‘sequence’  $ma \gg mb$  is a special case of ‘bind’ in which the computation  $mb$  does not depend on the  $A$  value returned by  $ma$ .

We work in the equational theory of the  $\lambda$ -calculus, as is common when discussing monads in Haskell; our presentation is a special case of the general categorical treatment of monads. The monad operations are required to satisfy the following three equational laws. The first two assert that *return* is a left and right unit for the ‘bind’ operation and the third that ‘bind’ is associative. (As usual,  $\lambda$ -binding scope extends as far to the right as possible. In the third equation,  $a$  is not free in  $g$ .)

$$\begin{aligned} \text{return } a \gg\equiv f & = f a \\ ma \gg\equiv \text{return} & = ma \\ ma \gg\equiv (\lambda a . (f a \gg\equiv g)) & = (ma \gg\equiv f) \gg\equiv g \end{aligned}$$

As a corollary, ‘sequential composition’ ( $\gg$ ) is associative, with left unit *return* ( $\lambda$ ).

*The State Monad.* A distinguished instance of the above concept is  $M_S$ , the *state monad on type S*, representing computations with access to a single updateable memory cell of type  $S$ . We define  $M_S A = S \rightarrow A \times S$  so that a computation of type  $A \rightarrow M_S B$  takes input  $a : A$ , and then can query the (old) state  $s : S$ , before return-

ing a new state  $s' : S$  and a result  $b : B$ . The monadic operations of  $M_S$  are defined below. The return operation takes a value  $a : A$  and produces a computation which, for any initial state  $s : S$ , returns the value  $a$  and leaves the state  $s$  untouched. The ‘bind’ operation  $\gg=$  chains together two stateful computations, using the final state  $s'$  of the first computation as the initial state of the second.

$$\begin{aligned} \text{return} & : A \rightarrow (S \rightarrow A \times S) \\ \text{return } a & = \lambda s . (a, s) \\ (\gg=) & : (S \rightarrow A \times S) \rightarrow (A \rightarrow (S \rightarrow B \times S)) \rightarrow (S \rightarrow B \times S) \\ ma \gg= f & = \lambda s . \mathbf{let} (a, s') = ma \mathbf{in} f a s' \end{aligned}$$

In addition to the generic operations `return` and `\gg=`, the state monad supports two operations `get`, `set`, to read and write the state:

$$\begin{aligned} \text{get} & : M_S S \\ \text{get} & = \lambda s . (s, s) \\ \text{set} & : S \rightarrow M_S () \\ \text{set } s' & = \lambda s . (((), s')) \end{aligned}$$

In general, one may characterise state monads with multiple memory cells in terms of an algebraic theory of reads and writes, with seven equations [4]. In the restricted setting of a single memory cell, the theory reduces to the following four equations:

$$\begin{aligned} \text{(GG)} \quad \text{get} \gg= \lambda s . \text{get} \gg= \lambda s' . k s s' & = \text{get} \gg= \lambda s . k s s \\ \text{(GS)} \quad \text{get} \gg= \text{set} & = \text{return } () \\ \text{(SG)} \quad \text{set } s \gg= \text{get} & = \text{set } s \gg= \text{return } s \\ \text{(SS)} \quad \text{set } s \gg= \text{set } s' & = \text{set } s' \end{aligned}$$

It is routine to verify that the above definitions of `get` and `set` satisfy these laws. However, in the algebraic perspective, one abstracts away from the specific concrete representation  $M_S$  and the corresponding implementations of `get` and `set`, and instead considers a ‘state monad on  $S$ ’ abstractly to be *any* monad  $M$  equipped with the additional structure of `get` and `set` satisfying the above four laws.

*Asymmetric lenses via the state monad.* An asymmetric lens [1] between  $S$  and  $V$  consists of a pair  $l$  of functions, usually called ‘get’ and ‘put’, which we write as follows:

$$\begin{aligned} l.\text{get} & : S \rightarrow V \\ l.\text{put} & : S \rightarrow V \rightarrow S \end{aligned}$$

The idea is that  $S$  and  $V$  represent *source* and *view* data, e.g. in a database;  $V$  is derived from  $S$  using  $l.\text{get}$ , and  $l.\text{put}$  computes a modified  $S$  on the basis of an old  $S$  and an updated  $V$ .

Given such a lens  $l$ , the state monad  $M_S$  admits computations  $\text{get}_l$ ,  $\text{set}_l$ , where  $\text{set}_l$  takes input from  $V$ , updates the state  $S$ , and returns void; and  $\text{get}_l$  is the trivially stateful operation that queries but doesn’t change the state  $S$ , and returns the  $V$  view of it:

$$\begin{aligned} \text{get}_l & : M_S V \\ \text{get}_l & = \lambda s . (l.\text{get } s, s) \\ \text{set}_l & : V \rightarrow M_S () \\ \text{set}_l v & = \lambda s . (((), l.\text{put } s v)) \end{aligned}$$

These computations do not allow us to observe, or update, the underlying state  $S$ , except via the view type  $V$ . But viewed as abstract operations relative to an arbitrary monad  $M$ , the structure

$$\begin{aligned} \text{get}_l & : M V \\ \text{set}_l & : V \rightarrow M () \end{aligned}$$

defines a state monad on  $V$ , provided that the equational laws hold.

In the special case of the *identity* lens  $l = \text{id}$ , between  $S$  and  $S$ , where  $\text{id}.\text{get}$  just reads the state, and  $\text{id}.\text{set}$  updates it, we have:

$$\begin{aligned} \text{get}_{\text{id}} & = \lambda s . (s, s) \\ \text{set}_{\text{id}} s' & = \lambda s . (((), s')) \end{aligned}$$

i.e. we obtain the state monad structure  $(M_S, \text{get}, \text{set})$  on  $S$ .

Thus, an asymmetric lens  $l$  gives rise to *two* distinct state monad structures, one on  $V$  derived from  $l$ , the other on  $S$  corresponding to the special case  $\text{id}$ . Each accesses the *same* underlying state; we say the two structures are *entangled*. In the rest of this paper, we consider such entangled state monads in general. The generalisation turns out to be both simple and powerful: several other bx formalisms are instances of this notion, corresponding to monads which present two updateable views of some shared, possibly hidden, state. In the next section we give details of the generalisation. We revisit the discussion of asymmetric (and other) lenses, in more detail, in Section 4.

### 3. ENTANGLED STATE MONADS

We now show that a monad that exhibits the structure of a state monad in two ways is essentially a bidirectional transformation. We do this by introducing two definitions, those of ‘set-bx’ (corresponding directly to state monads) and ‘put-bx’ (corresponding more closely to symmetric lenses) and showing that they are equivalent. (The proofs are included in an extended paper currently in preparation.) We use the umbrella term ‘entangled state monad’ for these two formulations.

#### 3.1 Set-bx

Given types  $A, B$ , we define a *set-bx between  $A$  and  $B$*  to be a monad  $M$ , equipped with four operations:

$$\begin{aligned} \text{get}_A & : M A \\ \text{get}_B & : M B \\ \text{set}_A & : A \rightarrow M () \\ \text{set}_B & : B \rightarrow M () \end{aligned}$$

that satisfy the three laws for  $\text{get}_A$  and  $\text{set}_A$

$$\begin{aligned} \text{(GG)} \quad \text{get}_A \gg= \lambda s . \text{get}_A \gg= \lambda s' . k s s' & = \text{get}_A \gg= \lambda s . k s s \\ \text{(GS)} \quad \text{get}_A \gg= \text{set}_A & = \text{return } () \\ \text{(SG)} \quad \text{set}_A a \gg= \text{get}_A & = \text{set}_A a \gg= \text{return } a \end{aligned}$$

and symmetrically for  $\text{get}_B$  and  $\text{set}_B$ . A set-bx that in addition satisfies the following:

$$\text{(SS)} \quad \text{set}_A a \gg= \text{set}_A a' = \text{set}_A a'$$

(and symmetrically in  $B$ ) is called *overwritable*.

We write  $(\text{get}_A, \text{get}_B, \text{set}_A, \text{set}_B) : A \xleftrightarrow{M} B$  to indicate that  $M$  is a set-bx between  $A$  and  $B$  equipped with operations  $\text{get}_A, \text{etc}$ . When discussing more than one such structure, we write  $t : A \xleftrightarrow{M} B$  and  $t.\text{get}_A$  and so on for the operations of  $t$ .

#### 3.2 Put-bx

Given types  $A, B$ , we define a *put-bx between  $A$  and  $B$*  to be a monad  $M$ , equipped with four operations:

$$\begin{aligned} \text{get}_A & : M A \\ \text{get}_B & : M B \\ \text{put}_A^B & : A \rightarrow M B \\ \text{put}_B^A & : B \rightarrow M A \end{aligned}$$

satisfying the following laws:

$$\begin{aligned} \text{(GG)} \quad \text{get} \gg= \lambda s . \text{get} \gg= \lambda s' . k s s' & = \text{get} \gg= \lambda s . k s s \\ \text{(GP)} \quad \text{get}_A \gg= \text{put}_A^B & = \text{get}_B \\ \text{(PG}_1) \quad \text{put}_A^B a \gg= \text{get}_A & = \text{put}_A^B a \gg= \text{return } a \\ \text{(PG}_2) \quad \text{put}_A^B a \gg= \text{get}_B & = \text{put}_A^B a \end{aligned}$$

(and symmetrically, swapping  $A$  and  $B$ ).



A put-bx that in addition satisfies the following:

$$(PP) \quad \text{put}_A^B a \gg \text{put}_A^B a' = \text{put}_A^B a'$$

(and symmetrically in  $B$ ) is called *overwriteable*.

As above, we write  $(\text{get}_A, \text{get}_B, \text{put}_A^B, \text{put}_B^A) : A \xleftrightarrow{M} B$  to indicate that  $M$  is a put-bx with operations  $\text{get}_A$ , etc., and write  $t : A \xleftrightarrow{M} B$ ,  $t.\text{get}_A$  and so on when discussing more than one such structure.

### 3.3 Relating set-bx and put-bx

We will show that set-bx and put-bx are equivalent in the following sense: for each set-bx  $t : A \xleftrightarrow{M} B$  we can construct a put-bx  $\text{set2pp}(t) : A \xleftrightarrow{M} B$  and for each put-bx  $u : A \xleftrightarrow{M} B$  we can construct a set-bx  $\text{pp2set}(u) : A \xleftrightarrow{M} B$ . Moreover, the two constructions are inverses:  $\text{pp2set}(\text{set2pp}(t)) = t$  and  $\text{set2pp}(\text{pp2set}(u)) = u$ . This means that any equation satisfied by all set-bx translates to an equation that holds for all put-bx, and vice versa. So, we can work with set-bx or put-bx as convenient, justifying our overloaded notation  $t : A \xleftrightarrow{M} B$ .

The translations are defined as follows. Given set-bx  $t : A \xleftrightarrow{M} B$ , define put-bx  $\text{set2pp}(t)$  by:

$$\begin{aligned} \text{set2pp}(t).\text{get}_A &= t.\text{get}_A \\ \text{set2pp}(t).\text{get}_B &= t.\text{get}_B \\ \text{set2pp}(t).\text{put}_A^B a &= t.\text{set}_A a \gg t.\text{get}_B \\ \text{set2pp}(t).\text{put}_B^A b &= t.\text{set}_B b \gg t.\text{get}_A \end{aligned}$$

Likewise, given put-bx  $u : A \xleftrightarrow{M} B$ , we define set-bx  $\text{pp2set}(u)$  as follows:

$$\begin{aligned} \text{pp2set}(u).\text{get}_A &= u.\text{get}_A \\ \text{pp2set}(u).\text{get}_B &= u.\text{get}_B \\ \text{pp2set}(u).\text{set}_A a &= u.\text{put}_A^B a \gg \text{return } () \\ \text{pp2set}(u).\text{set}_B b &= u.\text{put}_B^A b \gg \text{return } () \end{aligned}$$

LEMMA 1. *If  $t : A \xleftrightarrow{M} B$  is an (overwriteable) set-bx then  $\text{set2pp}(t) : A \xleftrightarrow{M} B$  is an (overwriteable) put-bx.*

LEMMA 2. *If  $u : A \xleftrightarrow{M} B$  is an (overwriteable) put-bx then  $\text{pp2set}(u) : A \xleftrightarrow{M} B$  is an (overwriteable) set-bx.*

LEMMA 3. *Translations  $\text{pp2set}(\cdot)$  and  $\text{set2pp}(\cdot)$  are inverses.*

### 3.4 Entanglement

Note that the state monad on pairs  $M_{A \times B}$  determines a set-bx, with

$$\begin{aligned} \text{get}_A &= \text{get} \gg \lambda(a, -) . \text{return } a \\ \text{get}_B &= \text{get} \gg \lambda(-, b) . \text{return } b \\ \text{set}_A a &= \text{get} \gg \lambda(-, b) . \text{set}(a, b) \\ \text{set}_B b &= \text{get} \gg \lambda(a, -) . \text{set}(a, b) \end{aligned}$$

However, this structure also satisfies stronger laws than our definitions require; in particular, commutativity of *sets*:

$$\text{set}_A a \gg \text{set}_B b = \text{set}_B b \gg \text{set}_A a$$

This law is not required of a set-bx; it is consistent with the set-bx laws that the  $A$  and  $B$  components of the state be ‘‘entangled’’, in the sense that setting one component also changes the other to restore consistency; in other words, that  $\text{set}_A$  and  $\text{set}_B$  need not commute. The monad  $M_{A \times B}$  arises simply as a special case of our general analysis of algebraic bx in Section 4 below, in which the consistency relation is universally true:  $\text{set}_A$  automatically restores consistency without the need to change  $B$  and vice versa.

## 4. INSTANCES

In this section we justify our view that set-bx (and hence also put-bx) structures are a general form of state-based bx, by showing how they capture the usual presentations such as asymmetric and symmetric lenses. Even though symmetric lenses subsume asymmetric lenses and algebraic bx, it is instructive to start with the simpler cases. We also give a simple example of a stateful bx that is not (isomorphic to) a symmetric lens. Investigation of other instances, and their relationships, is ongoing work.

*Asymmetric lenses.* Let  $l : A \rightleftharpoons B$  be a classic asymmetric lens, i.e.  $l.\text{get} : A \rightarrow B$  and  $l.\text{put} : A \rightarrow B \rightarrow A$ . We may construct a set-bx  $l : A \xleftrightarrow{M_A} B$  (where  $M_A$  is the state monad on state type  $A$ , as introduced in Section 2 above) as follows:

$$\begin{aligned} \text{get}_A &= \lambda a . (a, a) \\ \text{get}_B &= \lambda a . (l.\text{get } a, a) \\ \text{set}_A a' &= \lambda a . ((), a') \\ \text{set}_B b' &= \lambda a . ((), l.\text{put } a b') \end{aligned}$$

If  $l$  is a so-called ‘well-behaved’ lens, then it also satisfies:

$$\begin{aligned} (\text{GetPut}) \quad l.\text{put } a (l.\text{get } a) &= a \\ (\text{PutGet}) \quad l.\text{get } (l.\text{put } a b) &= b \end{aligned}$$

Finally, an asymmetric lens may optionally satisfy:

$$(\text{PutPut}) \quad l.\text{put } (l.\text{put } a b) b' = l.\text{put } a b'$$

in which case it is called *very well-behaved*.

LEMMA 4. *If the asymmetric lens  $l : A \rightleftharpoons B$  is well-behaved, then the above definitions indeed make  $l : A \xleftrightarrow{M_A} B$  into a set-bx. If  $l$  is very well-behaved, then  $l : A \xleftrightarrow{M_A} B$  is also overwriteable.*

*Algebraic bxs.* Let  $(R, \vec{R}, \overleftarrow{R})$  be an algebraic bx  $A \leftrightarrow B$  in the style of Stevens [5], i.e.,  $R \subseteq A \times B$ ,  $\vec{R} : A \times B \rightarrow B$ ,  $\overleftarrow{R} : A \times B \rightarrow A$ , satisfying the conditions

$$\begin{aligned} (\text{Correct}) \quad (a, \vec{R}(a, b)) &\in R \\ (\text{Hippocratic}) \quad R(a, b) &\Rightarrow \vec{R}(a, b) = b \end{aligned}$$

and symmetrically for  $\overleftarrow{R}$ . We say  $R$  is *undoable* if it also satisfies

$$(\text{Undoable}) \quad R(a, b) \Rightarrow \vec{R}(a, \vec{R}(a', b)) = b$$

and symmetrically for  $\overleftarrow{R}$ .

Let  $M_R$  be the state monad over  $R$ , viewing  $R$  as a set of pairs,  $R \subseteq A \times B$ . Then we define the following operations:

$$\begin{aligned} \text{get}_A &= \lambda(a, b) . (a, (a, b)) \\ \text{get}_B &= \lambda(a, b) . (b, (a, b)) \\ \text{set}_A a' &= \lambda(a, b) . ((), (a', \vec{R}(a', b))) \\ \text{set}_B b' &= \lambda(a, b) . ((), (\overleftarrow{R}(a, b'), b')) \end{aligned}$$

The condition (Correct) ensures that  $\text{set}_A a'$  and  $\text{set}_B b'$  are well-defined functions  $R \rightarrow () \times R$ , and thus preserve the consistency of pairs  $(a, b) \in R$ .

LEMMA 5. *For any algebraic bx  $(R, \vec{R}, \overleftarrow{R})$ , the above operations make  $M_R$  into a set-bx. If  $(R, \vec{R}, \overleftarrow{R})$  is undoable, then  $M_R$  is also overwriteable.*

*Symmetric lenses.* Let  $l : A \xleftrightarrow{C} B$  be a symmetric lens as presented by Hofmann et al. [2]. That is, let  $l = (\text{putl}, \text{putr})$  consist of a pair of functions

$$\begin{aligned} \text{putl} &: A \times C \rightarrow B \times C \\ \text{putr} &: B \times C \rightarrow A \times C \end{aligned}$$

which satisfy

$$\begin{aligned} (\text{PutRL}) \quad \text{putr}(a, c) = (b, c') &\Rightarrow \text{putl}(b, c') = (a, c') \\ (\text{PutLR}) \quad \text{putl}(b, c) = (a, c') &\Rightarrow \text{putr}(a, c') = (b, c') \end{aligned}$$

Let  $M_I$  be the state monad  $M_T$  over the set  $T$  of *consistent* states in  $A \times B \times C$ , i.e., those triples  $(a, b, c) \in A \times B \times C$  satisfying

$$\text{putr}(a, c) = (b, c) \quad \text{and} \quad \text{putl}(b, c) = (a, c)$$

Then define the following operations for  $M_I$ :

$$\begin{aligned} \text{get}_A &= \lambda(a, b, c). (a, (a, b, c)) \\ \text{get}_B &= \lambda(a, b, c). (b, (a, b, c)) \\ \text{put}_A^B a' &= \lambda(a, b, c). \mathbf{let} (b', c') = \text{putr}(a', c) \mathbf{in} (b', (a', b', c')) \\ \text{put}_B^A b' &= \lambda(a, b, c). \mathbf{let} (a', c') = \text{putl}(b', c) \mathbf{in} (a', (a', b', c')) \end{aligned}$$

We need to show that these operations are well defined in the sense that they preserve consistency of the state, and this is where we need the symmetric lens laws – once this is done, it is easy to see that these definitions satisfy the put-bx laws.

LEMMA 6. *Given any symmetric lens  $l = (\text{putl}, \text{putr}) : A \xleftrightarrow{C} B$ , the above operations are well-defined and make  $M_I$  into a put-bx.*

*Stateful bx.* We now consider an example that performs I/O side-effects, and thus by definition cannot be a symmetric lens (or any of the other bx mentioned above). We define a monad  $M$  that combines stateful updates (just on integer states, for simplicity) with Haskell-style monadic I/O; the latter is captured via a monad  $IO$  and an operation  $\text{print} : \text{String} \rightarrow IO ()$ . The return and  $\gg$  operations of  $M$  are therefore defined in terms of those of  $IO$ , so to be explicit we use subscripts below to disambiguate.

$$\begin{aligned} M A &= \text{Integer} \rightarrow IO (A, \text{Integer}) \\ \text{return}_M x &= \lambda s. \text{return}_{IO} (x, s) \\ ma \gg_M f &= \lambda s. ma \gg_{IO} \lambda(a, s'). f a s' \\ \text{get}_A &= \lambda s. \text{return}_{IO} (s, s) \\ \text{get}_B &= \lambda s. \text{return}_{IO} (s, s) \\ \text{set}_A a &= \lambda s. (\mathbf{if} a \neq s \\ &\quad \mathbf{then} \text{print} \text{ "Changed A"} \\ &\quad \mathbf{else} \text{return}_{IO} ()) \gg_{IO} \text{return}_{IO} ((), a) \\ \text{set}_B b &= \lambda s. (\mathbf{if} b \neq s \\ &\quad \mathbf{then} \text{print} \text{ "Changed B"} \\ &\quad \mathbf{else} \text{return}_{IO} ()) \gg_{IO} \text{return}_{IO} ((), b) \end{aligned}$$

That is, a computation in monad  $M$  yielding a result of type  $A$  amounts to an  $IO$ -computation yielding a pair of an  $A$  and a new  $Integer$  state, given as input an old  $Integer$  state. This is a set-bx: in particular, its behaviour satisfies the laws (GG), (GS) and (SG). Its *set* operations are side-effecting, but the side-effects only occur when the state is changed. For simplicity, we have taken the underlying bidirectional transformation to be trivial, but we should be able to add similar stateful behaviour to any (symmetric) lens or algebraic bx following a similar pattern.

## 5. CONCLUSIONS

Lenses are traditionally presented asymmetrically, whereas many bx applications such as model synchronisation are entirely symmetric. Symmetric lenses [2] and algebraic bx [5] cover the more general symmetric case, but both formulations go beyond equational logic. We have shown a very simple *equational* characterisation

that unifies lenses, symmetric lenses, and algebraic bx, by a natural generalisation of the ‘get’ and ‘set’ operations of the state monad. Interestingly, the notions of *consistency* for algebraic bx and *complement* disappear into the hidden state of the monad. We expect to be able to accommodate bx with richer complements or witness structures in the same way. Moreover, our approach offers the possibility of generalisation to reconcile effects such as I/O, non-determinism, exceptions, or probabilistic choice with bidirectionality, drawing on the rich theory of monads, and possibly leading to a theory of *bidirectional programming with effects*.

This is work in progress. We are currently investigating the central issues of *equivalence* and *composition* of entangled state monads. Symmetric lenses are quotiented by an equivalence relation in order for properties such as associativity of composition to hold. We expect something similar to be needed for entangled state monads. Indeed, the question of whether entangled state monads can be composed seems nontrivial; some restrictions on the class of monads considered may be necessary for composability.

We have considered entangled state monads only in relatively standard settings, such as the category of sets and functions (in the guise of Haskell types and functions). Another interesting direction may be to explore other settings, such as partial orders, metric spaces, or topologies, which may offer insights into notions of least change or predictable behaviour.

## Acknowledgements

We thank the participants at the Banff Bx workshop, and Benjamin Pierce, for helpful comments, as well as the anonymous reviewers for their generous and thoughtful remarks. The work is partly supported by EPSRC grants EP/K020218/1 and EP/K020919/1.

## 6. REFERENCES

- [1] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems*, 29(3):17, May 2007.
- [2] M. Hofmann, B. C. Pierce, and D. Wagner. Symmetric lenses. In *ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, Austin, Texas, Jan. 2011.
- [3] E. Moggi. Computational lambda-calculus and monads. In *LICS*, pages 14–23. IEEE Computer Society, 1989.
- [4] G. D. Plotkin and J. Power. Notions of computation determine monads. In *FoSSaCS*, pages 342–356, 2002.
- [5] P. Stevens. Bidirectional model transformations in QVT: Semantic issues and open questions. *Journal of Software and Systems Modeling (SoSyM)*, 9(1):7–20, 2010.
- [6] P. Wadler. Monads for functional programming. In J. Jeuring and E. Meijer, editors, *Advanced Functional Programming*, volume 925 of *Lecture Notes in Computer Science*, pages 24–52. Springer, 1995.

# Spans of lenses

Michael Johnson  
School of Mathematics and Computing  
Macquarie University

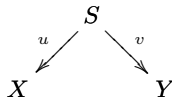
Robert Rosebrugh  
Department of Mathematics and Computer Science  
Mount Allison University

## ABSTRACT

Corresponding to the variety of notions of asymmetric lens, various notions of symmetric lens have been proposed. A common theory of the various asymmetric and symmetric lenses should result from a study of spans of asymmetric lenses. In order to define a category whose arrows are spans of asymmetric lenses, the fact that a cospan of asymmetric lenses may not have a pullback must be dealt with. In this article, after resolving that problem we develop the functors which exhibit a category whose arrows are spans of well-behaved lenses as a retract of a category whose arrows are the corresponding symmetric lenses. We relate them to the symmetric lenses of Hofmann, Pierce and Wagner.

## 1. INTRODUCTION

A *span* is a pair of functions, or more generally of arbitrary morphisms of a given kind, with common domain:



Such a span is often described as a “span from  $X$  to  $Y$ ”, and denoted  $u : X \leftarrow S \rightarrow Y : v$ . The object  $S$  is sometimes called the *peak* of the span and the arrows  $u$  and  $v$  are called the *legs* of the span.

Spans have been used in a variety of fields as diverse as the abstract theory of relations and the design of circuits and systems. Naturally the mathematical theory of spans is well-developed. Of particular importance: The composition of a span from  $X$  to  $Y$  and a span from  $Y$  to  $Z$  is a span from  $X$  to  $Z$  calculated by constructing a pullback, and two spans from  $X$  to  $Y$  are *span-equivalent* when there is an isomorphism between their peaks which commutes with their legs.

Spans of transformations arise widely in areas related to Bidirectional Transformations (Bx) too. Examples include model driven engineering, triple graph grammars, and sym-

metric lenses. However, in Bx the classical theory of spans is harder to apply. Even in the case of basic (asymmetric) lenses, the classical theory doesn’t apply because in the category whose morphisms are lenses, what one might expect to be the pullback of lenses need not satisfy the universal property of a pullback — the difficulty that arises is that the universally provided comparison morphism may not have a lens structure.

Even if we neglect for a moment the difficulty in properly defining composition of spans of lenses, there is a further complication: Span-equivalence does not seem to be the right notion of equivalence for spans of lenses. As Hofmann et al remarked in discussing spans of asymmetric lenses in [5] (full version), “in the span presentation there does not seem to be a natural and easy-to-use candidate for ... equivalence.”

The main goal of this paper is to develop the mathematical foundations required to support the use of spans in Bx. We show that

1. While categories whose morphisms are lenses may not have pullbacks, there is a framework that frequently allows us to work as if they did by finding canonical lens structures on the pullbacks of the Get functions.
2. While classical span-equivalence is far too strong a condition for equivalence of spans of lenses, there is a natural generalisation (replacing isomorphisms with suitably non-empty lenses) which seems right.

Combining these we have the mathematical foundations required. To demonstrate this we apply them here to the study of the symmetric lenses of [5], and in future work to a unified theory of symmetric lenses of many kinds.

A remark on the technical content. The paper is necessarily quite mathematical – we are after all building a mathematical foundation, and the measure of its utility or even correctness is its ability to provide precise proofs and insightful clarifications of mathematical results about spans of lenses. Many proofs have been omitted because, once the correct formulation has been found, the proof is relatively routine for those with category theoretic skills, and is unlikely to be very enlightening for those with less category theoretic experience. Other proofs have been sketched especially where an unusual approach might be required. The main contributions of this work are not the proofs themselves, but rather finding the right formulations (eg Proposition 5 for dealing with the missing pullbacks, and the equivalence  $\equiv_G$ ) which make the proofs feasible.

The structure of the paper is as follows. In Section 2 we review some details about asymmetric lenses and show how to canonically construct lenses on the pullbacks of the get functions of asymmetric lenses. The following section, Section 3, lays out the general theory that we use to systematically deal with these pullback like constructions (that aren't pullbacks) and introduces the equivalence relation which we will use on spans of asymmetric lenses. In Section 4 we introduce rl lenses — the approach to symmetric lenses that parallels that used in [5]. We show how to generalise that so that it can be applied in any category with products, making available immediately notions of symmetric lenses for graphs, categories, and ordered sets for example. In Section 5 we develop an equivalence for rl lenses motivated by, but different from, the equivalence for set based rl lenses in [5], and begin the development of two functors **A** and **S** which are used to compare span based and rl based symmetric lenses. Finally, in Section 6 we specialise to rl lenses with a pointed complement. These lenses correspond exactly to the lenses of [5] and we compare Hofmann et al's equivalence of such lenses with the equivalences proposed in this paper.

## 2. ASYMMETRIC LENSES

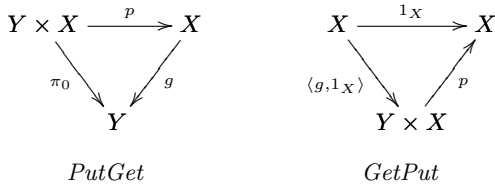
In this section we collect information about various notions of asymmetric lens along with some basic results that we will use later.

Let  $\mathbf{C}$  be a category with finite products. Categories such as the category of sets and functions, ordered sets and monotone mappings, and categories and functors, are some of the examples we have in mind. We recall the definition of asymmetric lens in  $\mathbf{C}$  [2, 5, 7].

**DEFINITION 1.** For objects  $X, Y$  in  $\mathbf{C}$ , an asymmetric lens in  $\mathbf{C}$  from  $X$  to  $Y$ , denoted  $L : X \rightarrow Y$  is  $L = (X, Y, g, p)$  where  $g : X \rightarrow Y$  is called the *Get morphism* and  $p : Y \times X \rightarrow X$  is called the *Put morphism*. A lens is called *well-behaved* if it satisfies:

- (i) (*PutGet*) the *Get* of a *Put* is the projection:  $gp = \pi_0$
- (ii) (*GetPut*) the *Put* for a trivially updated state is trivial:  $p(g, 1_X) = 1_X$

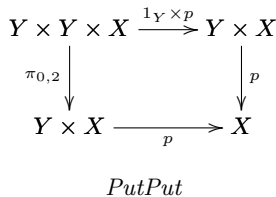
Diagrammatically, two commutative triangles:



A well-behaved lens is called *very well-behaved* if it satisfies:

- (iii) (*PutPut*) composing *Puts* does not depend on the first update:  $p(1_Y \times p) = p\pi_{0,2}$

Diagrammatically, a commutative square:



We showed in [7] that, up to isomorphism, a very well-behaved asymmetric lens  $L$  in  $\mathbf{C}$  has  $X = Y \times C$  for an object  $C$  of “complements” and then  $g$  is the projection  $g : Y \times C \rightarrow Y$  while the put is defined by  $p : Y \times (Y \times C) \rightarrow Y \times C = \pi_{0,2}$ , the projection onto the first and third factors. That generalises to categories with products (for example, those where the states of  $X$  and  $Y$  are, rather than mere sets, ordered sets or even more generally graphs or categories) the classical theory of constant complement updating [1]. Such lenses are algebras for a monad on  $\mathbf{C}/Y$ . Thus there is a well-defined notion of morphisms between lenses. However, our interest in this article is rather to treat various kinds of lenses as arrows of categories, so we will need a composition of lenses themselves.

Given lenses  $L = (X, Y, g_1, p_1)$  and  $M = (Y, Z, g_2, p_2)$ , the *composite lens* is  $ML = (X, Z, g, p)$  where  $g = g_2g_1$  and  $p$  is

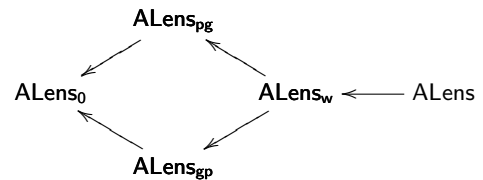
$$Z \times X \xrightarrow{1_Z \times (g_1, 1_X)} Z \times Y \times X \xrightarrow{p_2 \times 1_X} Y \times X \xrightarrow{p_1} X$$

or as an elementary formula:  $p(z, x) = p_1(p_2(z, g_1x), x)$ . For  $X$  in  $\mathbf{C}$  there is an identity lens  $1_X = (X, X, 1_X, p_0)$  with  $p_0$  the first projection,  $\pi_0$ . We will assume that products in  $\mathbf{C}$  are associative. With that assumption, composition of lenses in  $\mathbf{C}$  is associative and the identity lens acts as an identity.

The first additional properties of lenses are stable under composition:

**LEMMA 2.** Suppose that  $L, M$  and  $ML$  are defined as above. If  $L$  and  $M$  both satisfy *PutGet*, respectively *GetPut*, *PutPut*, then  $ML$  satisfies *PutGet*, respectively *GetPut* and, provided  $L$  satisfies *PutGet*, respectively *PutPut*.

Thus, there are categories we denote  $\mathbf{ALens}_0(\mathbf{C})$ ,  $\mathbf{ALens}_{pg}(\mathbf{C})$ ,  $\mathbf{ALens}_{gp}(\mathbf{C})$ ,  $\mathbf{ALens}_w(\mathbf{C})$  and  $\mathbf{ALens}(\mathbf{C})$  of asymmetric lenses, respectively lenses satisfying *PutGet*, *GetPut*, well-behaved lenses, and very well-behaved lenses in  $\mathbf{C}$ . In each case the objects are those of  $\mathbf{C}$  and the arrows are asymmetric lenses with the corresponding property. When  $\mathbf{C}$  is understood we will sometimes write  $\mathbf{ALens}_w$ , for example, for  $\mathbf{ALens}_w(\mathbf{C})$ . There are faithful, but evidently not full, inclusion functors



There are some further results about asymmetric lenses we need to record. First, finite product preserving functors preserve lens structures:

**PROPOSITION 3.** Suppose that  $\mathbf{C}$  and  $\mathbf{D}$  are categories with finite products and  $F : \mathbf{C} \rightarrow \mathbf{D}$  is a finite product preserving functor. If  $L = (X, Y, g, p)$  is an asymmetric lens in  $\mathbf{C}$ , and respectively satisfying *PutGet*, *GetPut*, a well-behaved lens or a very well-behaved lens, then  $FL = (FX, FY, Fg, Fp)$  is an (asymmetric) lens in  $\mathbf{D}$ , respectively satisfying *PutGet*, *GetPut*, a well-behaved lens, or a very well-behaved lens. If  $M$  is a lens in  $\mathbf{C}$  composable with  $L$ , then  $F(ML) = (FM)(FL)$ , and we obtain a functor, also denoted  $F$ ,

$$F : \mathbf{ALens}_0(\mathbf{C}) \rightarrow \mathbf{ALens}_0(\mathbf{D})$$

and respectively from  $\mathbf{ALens}_{\text{pg}}(\mathbf{C})$ ,  $\mathbf{ALens}_{\text{gp}}(\mathbf{C})$ ,  $\mathbf{ALens}_{\text{w}}(\mathbf{C})$  and  $\mathbf{ALens}(\mathbf{C})$ .

We denote the product-preserving inclusion of sets as discrete ordered sets by  $I$ , and of ordered sets as small categories by  $J$  in:

$$\mathbf{set} \xrightarrow{I} \mathbf{ord} \xrightarrow{J} \mathbf{cat}$$

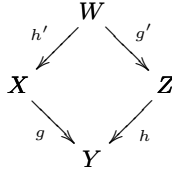
Thus there are faithful functors:

$$\mathbf{ALens}_0(\mathbf{set}) \xrightarrow{I} \mathbf{ALens}_0(\mathbf{ord}) \xrightarrow{J} \mathbf{ALens}_0(\mathbf{cat})$$

and similarly for the PutGet, GetPut, well-behaved and very well-behaved cases.

Next we consider pullbacks in  $\mathbf{C}$  and the various categories above. Remember that the lenses are *morphisms*. For example, when very well-behaved lenses are viewed as algebras for a monad on  $\mathbf{C}/Y$ , pullbacks exist in the algebras whenever they do in the base. Our case here is different. Since we need products and pullbacks in the base, we assume all finite limits.

**PROPOSITION 4.** *Suppose that  $\mathbf{C}$  is a category with finite limits and let  $L = (X, Y, g, p)$  be a morphism of  $\mathbf{ALens}_{\text{pg}}$ , resp.  $\mathbf{ALens}_{\text{w}}$  and  $\mathbf{ALens}$ , and  $h : Z \rightarrow Y$  be a morphism in  $\mathbf{C}$ . Let*



be a pullback in  $\mathbf{C}$ . Then  $L' = (W, Z, g', p')$  is in  $\mathbf{ALens}_{\text{pg}}$ , resp.  $\mathbf{ALens}_{\text{w}}$  and  $\mathbf{ALens}$ , where  $p' : Z \times W \rightarrow W$  is the unique morphism into the pullback determined by  $p(h \times h') : Z \times W \rightarrow X$  and  $\pi_0 : Z \times W \rightarrow Z$ . If also  $M = (Z, Y, h, q)$  is in  $\mathbf{ALens}_{\text{pg}}$ , resp.  $\mathbf{ALens}_{\text{w}}$  and  $\mathbf{ALens}$ , then for  $M' = (W, X, h', q')$  the corresponding lens we have  $LM' = ML'$ .

In the proof, the PutGet law gives  $gp(h \times h') = \pi_0(h \times h') = h\pi_0$  which is used to define  $p'$ . When PutGet is available, the remaining properties for  $L'$  follow routinely from those for  $L$ . When  $M'$  is defined, it is routine to check that the composite lenses are equal.

Note that we say nothing about  $\mathbf{ALens}_0$  nor  $\mathbf{ALens}_{\text{gp}}$ . It appears that at least the PutGet condition is needed both to define a sensible  $p'$  and to show that the square of lenses commutes.

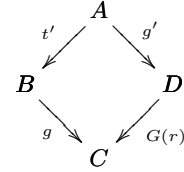
Finally, we make some comments regarding lenses involving initial objects. For any set  $Y$  there is a unique well-behaved asymmetric lens whose Get has empty domain in the category  $\mathbf{set}$  of sets and functions. The Put is the unique  $Y \times 0 \cong 0 \rightarrow 0$ . On the other hand, when the Get for a lens is a split epimorphism in  $\mathbf{C}$  then it has a section  $s$  satisfying  $gs = 1_Y$ . We call an asymmetric lens *split* if its Get is a split epimorphism. Thus, an asymmetric lens which is split and has an inhabited codomain also has an inhabited domain. If the codomain is not inhabited (for example, is empty in the  $\mathbf{set}$  case) then *its domain is also not inhabited*. In that case there is, of course, a unique lens structure taking the unique (identity) endo-function of the empty set for the Get.

### 3. CONSTRUCTION OF $\mathbf{SP}(G)$

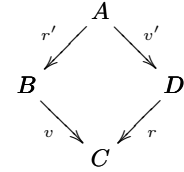
The various categories of asymmetric lenses usually do not have pullbacks. However, since the base category  $\mathbf{C}$  is assumed to have finite limits, a cospan of asymmetric lens Gets has a pullback in the *base* category  $\mathbf{C}$ . Moreover, the pullback projections in  $\mathbf{C}$  are themselves canonically the Gets of arrows (lenses) in the corresponding asymmetric lens category. We are interested in defining categories whose arrows are spans of the various sorts of asymmetric lenses. That motivates the following constructions.

Let  $\mathbf{C}$  be a category with finite limits and  $G : \mathbf{A} \rightarrow \mathbf{C}$  an identity on objects functor. The reader should think of  $\mathbf{A}$  as a category of asymmetric lenses and  $G$  as the forgetful functor which remembers only the Gets.

We suppose that there is an operation  $P$  on cospans of the form  $B \xrightarrow{g} C \xleftarrow{G(r)} D$  in  $\mathbf{C}$  which outputs an arrow  $P(g, r)$  of  $\mathbf{A}$ . We are thinking of  $r$  as an asymmetric lens and  $g$  as a  $\mathbf{C}$  arrow with the same codomain. We require three properties of  $P$ . Suppose first that  $G$  reflects isomorphisms, that is, if  $Gf$  is an isomorphism then  $f$  is an isomorphism. Next, we assume there is a pullback in  $\mathbf{C}$ :



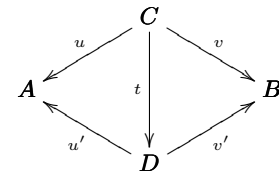
with  $t' = G(r')$  where  $r' = P(g, r)$  is in  $\mathbf{A}$ . Finally, if we also have that  $g = G(v)$  then for  $v' = P(G(r), v)$  the following square commutes in  $\mathbf{A}$ :



Note that the image under  $G$  of the square is still a pullback in  $\mathbf{C}$ .

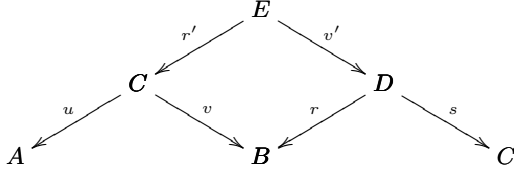
As noted above, the categories  $\mathbf{ALens}_{\text{pg}}(\mathbf{C})$ ,  $\mathbf{ALens}_{\text{w}}(\mathbf{C})$  and  $\mathbf{ALens}(\mathbf{C})$  of various types of asymmetric lenses all satisfy the hypotheses for  $G$  and  $P$ .

Given  $G$  and  $P$  as above, we define a category  $\mathbf{Sp}(G)$ . The objects of  $\mathbf{Sp}(G)$  are those of  $\mathbf{A}$  (or  $\mathbf{C}$ ). The arrows are equivalence classes for  $\equiv_G$  of spans in  $\mathbf{A}$ . The equivalence relation  $\equiv_G$  is generated by morphisms of spans in  $\mathbf{A}$  such as



in which  $u = u't$  and  $v = v't$  (this is what it means to be a morphism of spans) and for which  $G(t)$  is a split epimorphism. This condition on  $G(t)$  is important since  $G(t)$  will be a Get and we want to avoid cases where a lens whose Get has empty domain can make two spans equivalent. Two spans are equivalent, written  $(u, v) \equiv_G (u', v')$  if there is a

“zig-zag” (a string of length zero or more of arrows, adjacent members of which meet head to head or tail to tail) of the span morphisms described above between them. Composition in  $Sp(G)$  is defined by span composition in  $\mathbf{C}$  of representatives. That is, the composite of the  $\equiv_G$  equivalence classes of spans  $(u, v), (r, s)$  (in  $\mathbf{A}$ ) is the  $\equiv_G$  equivalence class of the span  $(ur', sv')$  where  $r' = P(G(v), r)$  and  $v' = P(G(r), v)$ , as in the diagram in  $\mathbf{A}$ :



PROPOSITION 5. *With the data just defined,  $Sp(G)$  is a category.*

The proof is largely routine but we do note that it uses the fact that a split epimorphism “pulls back” to (that is, it has as an opposite pullback projection) a split epimorphism.

#### 4. SYMMETRIC LENSES

As for asymmetric lenses, symmetric lenses were first defined in  $\mathbf{set}$  and the concept can be generalized. The asymmetric lens concept is expressible in any category with finite products. The same is true for the symmetric lenses (in  $\mathbf{set}$ ) of Hofmann, Pierce and Wagner [5].

DEFINITION 6. *Let  $\mathbf{C}$  be a category with finite products (including the empty product). For objects  $X, Y$  in  $\mathbf{C}$ , an rl lens from  $X$  to  $Y$  is a quintuple,  $L = (X, Y, C, r, l)$  where  $C$  is an object of  $\mathbf{C}$  of “complements” and  $r$  and  $l$  are morphisms*

$$r : X \times C \longrightarrow Y \times C \quad \text{and} \quad l : Y \times C \longrightarrow X \times C$$

satisfying the four equations

$$\pi_X l r = \pi_X : X \times C \longrightarrow X; \pi_C l r = \pi_C r : X \times C \longrightarrow C \quad (\text{PUTRL})$$

and

$$\pi_Y r l = \pi_Y : Y \times C \longrightarrow Y; \pi_C r l = \pi_C l : Y \times C \longrightarrow C \quad (\text{PUTLR})$$

An rl lens with a specified point  $m : 1 \longrightarrow C$  ( $m$  is for “missing”) in its complements is called a pointed-complement- or pc-symmetric lens and denoted  $L = (X, Y, C, r, l, m)$ . If  $L$  is an rl lens there is an opposite lens  $L^{op} = (Y, X, C, l, r)$  with the same complements and  $l$  and  $r$  interchanged.

REMARKS. A pc-symmetric lens in  $\mathbf{set}$  is what was called a symmetric lens in [5]. We will have more to say about them below, but for now we just note that the point  $m$  is not involved in the equations so it just ensures that  $C$  is inhabited.

For an rl lens in  $\mathbf{set}$ , the equations in the definition are equivalent to the implications used to define a symmetric lens in [5]. Those implications are

$$r(x, c) = (y, c') \Rightarrow l(y, c') = (x, c')$$

and

$$l(y, c) = (x, c') \Rightarrow r(x, c') = (y, c)$$

First, assume the equations in the definition and  $r(x, c) = (y, c')$ . We need to show that  $l(y, c') = (x, c')$ . Now since

$l(y, c') = lr(x, c)$  we have  $\pi_C l(y, c') = \pi_C l r(x, c) = \pi_C r(x, c) = \pi_C(y, c') = c'$  using the second equation of (PUTRL). Similarly  $\pi_X l(y, c') = \pi_X l r(x, c) = \pi_X(x, c) = x$  using the first (PUTRL) equation, so  $l(y, c') = (x, c')$ . The other implication follows similarly. Now assume the implications. The first shows that  $lr(x, c) = (x, c')$  where  $r(x, c) = (y, c')$ . Projections from this equation are exactly the (PUTRL) equations in the definition. The (PUTLR) equations follow from the second implication.

We have the following:

PROPOSITION 7. *For an rl lens  $L = (X, Y, C, r, l)$  in  $\mathbf{C}$ , the equations  $rlr = r$  and  $lrl = l$  hold.*

PROOF. Since  $\pi_X l r = \pi_X$ , we have  $\pi_X l r l = \pi_X l : Y \times C \longrightarrow X$  and  $\pi_C l r = \pi_C r$  implies  $\pi_C l r l = \pi_C r l = \pi_C l$  using the fourth equation. Since  $l r l$  and  $l$  have the same projections to  $X$  and  $C$ , they are equal. That  $rlr = r$  is similar.  $\square$

It appears that the equations in the preceding Proposition do not imply the rl lens equations.

For any  $L : X \longrightarrow Y$  in  $\mathbf{ALens}_w(\mathbf{set})$ , say  $L = (X, Y, g, p)$ , which is split by  $s : Y \longrightarrow X$ , there is a pc-symmetric lens  $T(L) = (X, Y, C, r, l, m)$  where  $C = \{f : Y \longrightarrow X \mid gf = 1_Y\}$  is the set of sections of  $g$ ,  $r(x, f) = (g(x), p(-, x))$ ,  $l(y, f) = (f(y), p(-, f(y)))$ , and  $m = s$ .

PROPOSITION 8. [5] *For an asymmetric lens  $L$  in  $\mathbf{ALens}_w(\mathbf{set})$  which is split,  $T(L)$  is a pc-symmetric lens in  $\mathbf{set}$  from  $X$  to  $Y$ .  $\square$*

We are more interested in relating rl lenses with spans of asymmetric lenses.

We begin by supposing that  $L = (X, Y, C, r, l)$  is an rl lens in  $\mathbf{C}$ . If  $L$  is an rl lens in  $\mathbf{set}$  it is easy to see that  $\{(x, y, c) \mid r(x, c) = (y, c)\} = \{(x, y, c) \mid l(y, c) = (x, c)\}$ . More generally,

PROPOSITION 9. *If  $e : S \longrightarrow X \times Y \times C$  is an equalizer of*

$$r\pi_{0,2}, \pi_{1,2} : X \times Y \times C \longrightarrow Y \times C$$

*then  $e : S \longrightarrow X \times Y \times C$  is an equalizer of  $l\pi_{1,2}, \pi_{0,2} : X \times Y \times C \longrightarrow X \times C$ .*

PROOF. Consider the diagram:

$$\begin{array}{ccccc} S' & \xrightarrow{e'} & X \times Y \times C & \xrightarrow{\pi_{0,2}} & X \times C & \xrightarrow{\pi_C} & C \\ & & & & & \searrow \pi_C & \\ & & & & & & Y \times C \\ & & & & & \swarrow \pi_Y & \\ & & & & & & Y \\ & & & & & \swarrow \pi_Y & \\ & & & & & & Y \end{array}$$

If  $e' : S' \longrightarrow X \times Y \times C$  is an equalizer of  $l\pi_{1,2}$  and  $\pi_{0,2}$  then it equalizes  $rl\pi_{1,2}$  and  $r\pi_{0,2}$ . Also,  $\pi_Y r l = \pi_Y$  by assumption, so  $r\pi_{0,2}e = \pi_{1,2}e$  and  $e'$  factors through the equalizer  $e : S \longrightarrow X \times Y \times C$  of  $r\pi_{0,2} = \pi_{1,2}$ . Similarly,  $e$  factors through  $e'$ , so  $e$  is also an equalizer of  $l\pi_{1,2}$  and  $\pi_{0,2}$ .  $\square$

The equalizer  $S$  is called the *consistent triples*. In  $\mathbf{set}$ , if  $(x, y, c)$  is a consistent triple for an rl lens then  $r(x, c) = (y, c)$  and  $l(y, c) = (x, c)$ . The definition of  $S$  as an equalizer extends the concept of consistent triples to other categories with products.

Furthermore the construction of  $S$  allows us to define a span of well-behaved asymmetric lenses from an rl lens as follows.

PROPOSITION 10. *Suppose that  $L = (X, Y, C, r, l)$  is an rl lens in  $\mathbf{C}$ . Let  $e : S_L \rightarrow X \times Y \times C$  be an equalizer of  $r\pi_{0,2}$  and  $\pi_{1,2}$ . Then there is a span*

$$L_l : X \leftarrow S_L \rightarrow Y : L_r$$

in  $\mathbf{ALens}_w$  from  $X$  to  $Y$  with Gets defined by  $g_l = \pi_{X,e}$ ,  $g_r = \pi_{Y,e}$ . The Put for  $L_l$  is defined by

$$p_l : X \times S_L \xrightarrow{1_X \times e} X \times X \times Y \times C \xrightarrow{\pi_{0,3}} X \times C \\ \xrightarrow{\Delta_X \times 1_C} X \times X \times C \xrightarrow{1_X \times r} S_L$$

The Put for  $L_r$  is similar.

We denote the span  $(L_l, L_r)$  by  $A(L)$ . In **set** the formula for the Put for  $L_l$  is  $p_l(x', (x, y, c)) = (x', r(x', c))$ .

Thus to every rl lens  $L$  we have associated a span of asymmetric well behaved lenses  $A(L)$ . Indeed the main purpose of this paper is to develop the machinery to allow us to, given a category of asymmetric lenses like  $\mathbf{ALens}_w$ , and its forgetful functor that remembers only the Gets, use the  $Sp$  construction to obtain the corresponding category of symmetric lenses. In future work we apply this construction to a range of types of lenses including delta lenses [3, 4], c-lenses [8], very well behaved lenses, and so on, to obtain categories of symmetric lenses of each kind.

Returning to our case in point, denote by  $U_w$  the forgetful functor from  $\mathbf{ALens}_w$  to  $\mathbf{C}$  which remembers only the Gets. The **category of symmetric well behaved lenses in  $\mathbf{C}$**  is defined to be  $Sp(U_w)$  and is denoted  $\mathbf{SLens}_w$ .

Of course, we should compare in more detail the symmetric well behaved lenses in  $\mathbf{C}$  with the rl lenses in  $\mathbf{C}$ . To do that properly we need to define composites of rl lenses, and an appropriate equivalence of rl lenses.

## 5. EQUIVALENCE OF RL LENSES

For rl lenses  $L_1 = (X, Y, C_1, r_1, l_1)$  and  $L_2 = (X, Y, C_2, r_2, l_2)$  we introduce a relation  $R$  and say that  $L_1 R L_2$  if there is a well-behaved asymmetric lens  $L = (C_1, C_2, t, p)$  from  $C_1$  to  $C_2$  with  $t$  a split epimorphism and such that  $L$  respects the operations of  $L_1$  and  $L_2$ . That is,

$$r_2(X \times t) = (Y \times t)r_1 \text{ and } l_2(Y \times t) = (X \times t)l_1$$

and

$$r_1(X \times p) = (Y \times p)(r_2 \times C_1) \text{ and } l_1(Y \times p) = (X \times p)(l_2 \times C_1).$$

(In other words  $L$  commutes with the rl structures.) The relation  $R$  on rl lenses from  $X$  to  $Y$  generates an equivalence relation (its reflexive, symmetric, transitive closure) denoted  $\equiv_{rl}$  on rl lenses from  $X$  to  $Y$ . We denote the  $\equiv_{rl}$  equivalence class of  $L = (X, Y, C, r, l)$  by  $[L]_{rl}$ .

DEFINITION 11. *For rl lenses  $L = (X, Y, C, r, l)$  and  $M = (Y, Z, C', r', l')$  the rl-composite lens is*

$$LM = (X, Z, C'', r'', l'', m'')$$

where  $C'' = C \times C'$ ,

$$r'' = \langle \pi_{0,2}, \pi_1 \rangle (r' \times 1_C) \langle \pi_{0,2}, \pi_1 \rangle (r \times 1_{C'}),$$

and

$$l'' = (l \times 1_{C'}) \langle \pi_{0,2}, \pi_1 \rangle (l' \times 1_C) \langle \pi_{0,2}, \pi_1 \rangle.$$

PROPOSITION 12. *For rl lenses  $L_1, L_2$  from  $X$  to  $Y$  and  $M$  from  $Y$  to  $Z$  in  $\mathbf{C}$ , the rl-composite lens  $ML_1$  is an rl lens from  $X$  to  $Z$ . Moreover, if  $L_1 \equiv_{rl} L_2$  then  $ML_1 \equiv_{rl} ML_2$ .*

PROOF. The first statement is similar to that in [5] using a twist isomorphism. For the second point, it is straightforward to prove the statement for the relation  $R$  and then it follows for  $\equiv_{rl}$ .  $\square$

COROLLARY 13. *For rl lenses  $L_1, L_2$  from  $X$  to  $Y$  and  $M_1, M_2$  from  $Y$  to  $Z$  in  $\mathbf{C}$ , if  $L_1 \equiv_{rl} L_2$  and  $M_1 \equiv_{rl} M_2$  then  $M_1 L_1 \equiv_{rl} M_2 L_2$ .*

The corollary allows us to define an associative composition on  $\equiv_{rl}$  classes. The proof of associativity is again like that in [5]. We can now define the **category of rl lenses,  $\mathbf{RLens}$** . The objects are those of  $\mathbf{C}$ . The arrows from  $X$  to  $Y$  are the  $\equiv_{rl}$  classes of rl lenses from  $X$  to  $Y$  with the composition just described.

PROPOSITION 14. *There is an identity on objects functor we call  $\mathbf{A} : \mathbf{RLens} \rightarrow \mathbf{SLens}_w$  defined by  $\mathbf{A}([L]_{rl}) = [A(L)]_{U_w}$ .*

PROOF. The proof proceeds by showing that  $\mathbf{A}$  is well-defined, independently of the choice of representative of the rl equivalence class  $[L]_{rl}$ , and that the composite of rl lenses is sent, up to  $\equiv_{U_w}$ , to the composite of the corresponding spans in  $Sp(U_w)$ .  $\square$

Given a span  $L : X \leftarrow S \rightarrow Y : M$  in  $\mathbf{ALens}_w$  (that is a representative for an equivalence class which is a morphism in  $\mathbf{SLens}_w$ ) we determine an rl lens denoted  $S(L, M)$  as follows.

PROPOSITION 15. *Suppose that  $L = (S, X, g_l, p_l)$  and that  $M = (S, Y, g_r, p_r)$  form a span of well-behaved asymmetric lenses in  $\mathbf{C}$ . Define*

$$r = \langle g_r, 1 \rangle p_l : X \times S \rightarrow Y \times S$$

and

$$l = \langle g_l, 1 \rangle p_r : Y \times S \rightarrow X \times S$$

then  $S(L, M) = (X, Y, S, r, l)$  is an rl lens in  $\mathbf{C}$ .

PROOF. The proof is a routine verification that  $r$  and  $l$  so defined satisfy the four equations called PUTRL and PUTLR in Definition 6.  $\square$

PROPOSITION 16. *Suppose that  $L : X \leftarrow S \rightarrow Y : M$  is a span of well-behaved asymmetric lenses in  $\mathbf{C}$  and denote  $AS(L, M)$  by  $L_l : X \leftarrow S_L \rightarrow Y : L_r$ . There is an isomorphism  $g : S \rightarrow S_L$  that is a morphism of spans, and consequently  $AS(L, M) \equiv_{U_w} (L, M)$ , and they are isomorphic as spans of asymmetric lenses.*

PROOF. The main point in the proof is to show that the consistent triples  $S_{S(L, M)}$  have, in the case when  $\mathbf{C}$  is the

category of sets, the form  $(g_l(s), g_r(s), s)$  where  $g_l$  and  $g_r$  are the gets of the asymmetric lenses  $L$  and  $M$  respectively. The required isomorphism  $g$  is then apparent, being an isomorphism it has a canonical lens structure, and as a lens it commutes with the four asymmetric lenses  $L, M, L_l$  and  $L_r$ .  $\square$

PROPOSITION 17. *Suppose that  $L : X \leftarrow S \rightarrow Y : M$  and  $L' : X \leftarrow S' \rightarrow Y : M'$  are  $\equiv_{U_w}$  equivalent spans of well behaved asymmetric lenses in  $\mathbf{C}$ . Then  $S(L, M) \equiv_{r_l} S(L', M')$  and so  $\mathbf{S}([(L, M)]_{\equiv_{U_w}}) = [S(L, M)]_{r_l}$  defines a functor  $\mathbf{S} : \mathbf{SLens}_w \rightarrow \mathbf{RLens}$ .*

PROOF. For the first point,  $S(L, M) \equiv_{r_l} S(L', M')$  provided that  $AS(L, M) \equiv_{U_w} AS(L', M')$ , but certainly we have  $AS(L, M) \equiv_{U_w} (L, M) \equiv_{U_w} (L', M') \equiv_{U_w} AS(L', M')$ .

To see that  $\mathbf{S}$  is a functor, we note first that it is identity on objects. The first part of the proposition shows that  $\mathbf{S}$  is well-defined on morphisms. Finally, to see that  $\mathbf{S}$  is compatible with composition it suffices to trace how composition works on each side, noting that Proposition 15 defines each of the  $r$  and  $l$  operations in terms of the lenses  $L$  and  $M$ .  $\square$

THEOREM 18.  $\mathbf{SLens}_w$  is a retraction of  $\mathbf{RLens}$  via the functors  $\mathbf{A}$  and  $\mathbf{S}$ .

PROOF. Both  $\mathbf{A}$  and  $\mathbf{S}$  are identity on objects.

If  $L : X \leftarrow S \rightarrow Y : M$  is a span of well-behaved asymmetric lenses in  $\mathbf{C}$ , then  $\mathbf{AS}([(L, M)]_{\equiv_{U_w}}) = [(L, M)]_{\equiv_{U_w}}$ .  $\square$

This result shows that every  $rl$  lens can be normalised into one of the form  $\mathbf{S}([(L, M)])$  for a span of asymmetric well behaved lenses  $L : X \leftarrow S \rightarrow Y : M$ , and that the category of  $rl$  lenses of that form is equivalent to the category  $\mathbf{SLens}_w$ .

## 6. PC-SYMMETRIC LENSES

There is a category of pc-symmetric lenses defined in [5] which we now review. Our goal is to relate the equivalence of symmetric well behaved lenses defined as spans,  $\equiv_{U_w}$ , with the equivalence of pc-symmetric lenses used in [5].

The main difficulty we face is that the pointing turns out to be fundamentally important to the equivalence of pc-symmetric lenses, although it plays no role in the algebraic structure, and so has been excluded from the definition of  $rl$  lenses and symmetric well behaved lenses.

DEFINITION 19. *For pc-symmetric lenses  $L = (X, Y, C, r, l, m)$  and  $M = (Y, Z, C', r', l', m')$  the pc-composite lens is*

$$L \circ M = (X, Z, C'', r'', l'', m'')$$

where  $C'' = C \times C'$ ,

$$r'' = \langle \pi_{0,2}, \pi_1 \rangle (r' \times 1_C) \langle \pi_{0,2}, \pi_1 \rangle (r \times 1_{C'}),$$

$$l'' = (l \times 1_{C'}) \langle \pi_{0,2}, \pi_1 \rangle (l' \times 1_C) \langle \pi_{0,2}, \pi_1 \rangle$$

and  $m'' = (m, m')$ . We display  $r''$  ( $l''$  is similar):

$$\begin{aligned} X \times C \times C' &\xrightarrow{r \times 1_{C'}} Y \times C \times C' \xrightarrow{\langle \pi_{0,2}, \pi_1 \rangle} Y \times C' \times C \\ &\xrightarrow{r' \times 1_C} Z \times C' \times C \xrightarrow{\langle \pi_{0,2}, \pi_1 \rangle} Z \times C \times C' \end{aligned}$$

PROPOSITION 20. *For pc-symmetric lenses  $L$  and  $M$  in  $\mathbf{C}$ , the pc-composite lens  $L \circ M$  is a pc-symmetric lens from  $X$  to  $Z$ .*

The proof for symmetric lenses in  $\mathbf{set}$  is in [5], and is similar for symmetric lenses in  $\mathbf{C}$ .

In [5] an equivalence relation on pc-symmetric lenses from  $X$  to  $Y$  in  $\mathbf{set}$  is defined as follows.

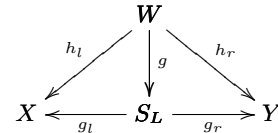
DEFINITION 21. *Suppose  $L_1 = (X, Y, C_1, r_1, l_1, m_1)$  and  $L_2 = (X, Y, C_2, r_2, l_2, m_2)$  are pc-symmetric lenses from  $X$  to  $Y$  and  $R \subseteq C_1 \times C_2$ . Say  $L_1$  is  $(R)$ -equivalent to  $L_2$ , denoted  $L_1 \equiv_R L_2$  iff  $(m_1, m_2)$  is in  $R$  and furthermore  $(c_1, c_2) \in R, x \in X, (y_1, c_3) = r_1(x, c_1), (y_2, c_4) = r_2(x, c_2)$  imply that  $y_1 = y_2$  and  $(c_3, c_4) \in R$  (1) and corresponding symmetric statements hold for  $l_1, l_2$ . We write  $L_1 \equiv_{pc} L_2$  if there exists an  $R$  such that  $L_1 \equiv_R L_2$*

Recall from [5] that  $\equiv_{pc}$  is an equivalence relation which respects pc-symmetric lens composition. There is a category  $\mathbf{pcLens}$  (called  $\mathbf{Lens}$  in [5]) whose arrows are  $\equiv_{pc}$  equivalence classes of pc-symmetric lenses with composition given by the pc-composite of representatives.

We want to relate the equivalence of spans of asymmetric lenses by  $\equiv_{U_w}$  with equivalence between pc-symmetric lenses given by  $\equiv_{pc}$ . We start with a generator of  $\equiv_{U_w}$ . We will show that when two symmetric well behaved lenses are each pointed compatibly (choose any point in the peak of one, and use the equivalence to determine the corresponding point in the other) then their corresponding pc-symmetric lenses are  $\equiv_{pc}$  equivalent.

PROPOSITION 22. *Suppose that  $L = (X, Y, C, r, l)$  is an  $rl$  lens with  $A(L)$  the span  $L_l : X \leftarrow S_L \rightarrow Y : L_r$  of well-behaved asymmetric lenses. Suppose that  $M = (W, S_L, g, p)$  is in  $\mathbf{ALens}_w$  and is split by  $s : S_L \rightarrow W$ . Define  $h_l = g_l g, h_r = g_r g, q_l : X \times W \rightarrow W$  by  $q_l(x, w) = p(p_l(x, gw), w)$  and  $q_r : Y \times W \rightarrow W$  by  $q_r(x, w) = p(p_r(x, gw), w)$ , defining asymmetric lenses  $M_l : X \leftarrow W \rightarrow Y : M_r$ . Define  $R = \{(w, \pi_C gw)\} \subseteq W \times C$ , and suppose  $m : 1 \rightarrow W$  is a point of  $W$  and hence  $(m, \pi_C gm)$  is in  $R$ . Let  $L_{gm} = (X, Y, C, r, l, \pi_C g(m))$  be the pc-symmetric lens pointed by  $\pi_C gm$ . Let  $S(M_l, M_r) = (X, Y, W, r_M, l_M)$  be the  $rl$  lens determined above and  $L_m = (X, Y, W, r_M, l_M, m)$  its corresponding pc-symmetric lens. Then  $L_m \equiv_R L_{gm}$ .*

PROOF. The situation is summed up in the following:



We first note that  $r_M : X \times W \rightarrow Y \times W$  is defined by  $r_M(x, w) = (h_r q_l(x, w), q_l(x, w))$ .

We are going to prove only implication (1), so suppose  $(w, \pi_C gw) \in R, x \in X, (y_1, w_1) = r_M(x, w)$  and  $(y_2, c_2) = r(x, \pi_C gw)$ . We need to show that  $y_1 = y_2$  and  $(w_1, c_2) \in R$ .



First

$$\begin{aligned}
y_1 &= h_r q_l(x, w) \\
&= g_r g_l q_l(x, w) \\
&= g_r g_l p_l(x, gw, w) \quad \text{by def of } q_l \\
&= g_r p_l(x, gw) \quad \text{by PutGet} \\
&= g_r(x, r(x, \pi_C gw)) \quad \text{def of } p_l \\
&= \pi_Y(r(x, \pi_C gw)) = y_2
\end{aligned}$$

Next  $w_1 = q_l(x, w)$  by definition, so we want to show that  $c_2 = \pi_C g(q_l(x, w))$  to get  $(w_1, c_2) \in R$ . Now  $c_2 = \pi_C(r(x, \pi_C gw))$ , so

$$\begin{aligned}
c_2 &= \pi_C(r(x, \pi_C gw)) \\
&= \pi_C \pi_{Y,C} p_l(x, gw) \quad \text{def of } p_l \\
&= \pi_C p_l(x, gw) \quad \text{comp'n of projections} \\
&= \pi_C g_l p_l(x, gw, w) \quad \text{by PutGet} \\
&= \pi_C g_l q_l(x, w) \quad \text{def of } q_l
\end{aligned}$$

That completes the proof.  $\square$

**COROLLARY 23.** *If  $L : X \leftarrow S \rightarrow Y : M$  and  $L' : X \leftarrow S' \rightarrow Y : M'$  are  $\equiv_{U_w}$  equivalent spans in  $\mathbf{SLens}_w$  via  $g : S \rightarrow S'$ ,  $m : 1 \rightarrow S$  and  $m' : 1 \rightarrow S'$  where  $m$  and  $m' = g(m)$  are related by  $\equiv_{U_w}$ . Then for the corresponding pc-symmetric lenses we have  $L_{m'} \equiv_{pc} L_m$ .*

Since we have it on generators of the equivalence, the corollary shows that  $\equiv_{U_w}$ , appropriately pointed to permit a comparison, is a finer equivalence relation than  $\equiv_{pc}$ .

## 7. FUTURE WORK

This paper has been about developing the theory of spans of lenses and an appropriate equivalence relation for them which can then be used *parametrically* in the type of asymmetric lens of interest.

Having laid the mathematical foundations for a theory of symmetric lenses of type X as spans of asymmetric lenses of type X we are in a position to define new symmetric lenses, and to make detailed comparisons with the various extant notions of symmetric lenses.

We have currently carried through this project for the delta lenses of Diskin et al [3, 4, 6], and we are progressing well with c-lenses [8]. The overall goal is a unified treatment of all the various kinds of symmetric lenses that have been defined, along with guidance for the future development of new types of symmetric lenses.

In each case, we strive, as in the preceding section of this paper, to relate the work to extant structures, and to use the theory to shed more light on why they are the way that they are, or to reveal when there are choices that have been made that could have been made differently (thus for example,  $\equiv_{pc}$  seems to be a coarser equivalence than is strictly needed, and we are studying this further).

In other future work we are exploring the lens-like aspects of our construction of  $Sp(G)$ . The operation  $P(g, r)$  can itself be seen as a kind of put operation which satisfies its own PutGet, GetPut and PutPut conditions. It appears that if they are formulated correctly the resultant structure is a c-lens — further evidence of the utility of bidirectional transformations (even in the development of the theory of bidirectional transformations!).

## 8. CONCLUSIONS

We have presented foundations for a consistent and unified treatment of several different types of asymmetric and symmetric lenses. In that treatment,  $Sp(U)$  is used to build categories of spans of asymmetric lenses.  $Sp(U)$  provides a different treatment to the usual treatment for composing spans, thus overcoming the difficulty about categories of lenses frequently not having pullbacks. In addition,  $Sp(U)$  uses a different equivalence from the one usually used in categories of spans. The new equivalence seems to be the natural generalisation for Bx as it replaces a requirement for an isomorphism between spans (a particularly restrictive kind of bidirectional transformation) with a lens.

The work has allowed us to compare the equivalence of the general theory with the equivalence proposed in [5]. The latter has some surprising features. It certainly achieves the outcomes its authors required, but it appears that the new equivalence can do the same without forcing so many different lenses to be equivalent.

## 9. ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of the Australian Research Council and the Natural Sciences and Engineering Research Council of Canada. In addition the authors are grateful for the valuable suggestions and thoughtful analysis of the anonymous referees.

## 10. REFERENCES

- [1] Bancilhon, F. and Spyratos, N. (1981) Update semantics of relational views, *ACM Trans. Database Syst.* **6**, 557–575.
- [2] Bohannon, A., Vaughan, J. and Pierce, B. (2006) Relational Lenses: A language for updatable views. *Proceedings of Principles of Database Systems (PODS) 2006*.
- [3] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki (2011), From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case, *Journal of Object Technology* **10**, 6:1–25, doi:10.5381/jot.2011.10.1.a6
- [4] Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki, Hartmut Ehrig, Frank Hermann and Francesco Orejas (2011), From State- to Delta-Based Bidirectional Model Transformations: the Symmetric Case, ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems: Springer, 10/2011.
- [5] Hofmann, M., Pierce, B., and Wagner, D. (2011) Symmetric Lenses. In ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), Austin, Texas.
- [6] Johnson, M. and Rosebrugh, R. (2013) Delta lenses and fibrations. to appear in *Electronic Communications of the EASST*, Proceedings of BX 2013.
- [7] Johnson, M., Rosebrugh, R. and Wood, R. J. (2010) Algebras and Update Strategies. *J.UCS* **16**, 729–748.
- [8] Johnson, M., Rosebrugh, R. and Wood, R. J. (2012) Lenses, fibrations and universal translations. *Mathematical Structures in Computer Science.* **22**, 25–42.

# Energy Data Management (EnDM)

Torben Bach Pedersen (Aalborg University, Denmark)

# Pipeline Production Data Model

Jitao Yang  
Production System Research  
Lab, CPPEI  
PetroChina Company Limited  
Beijing 102206, China  
yangjitao@petrochina.  
com.cn

Yu Fan  
Production System Research  
Lab, CPPEI  
PetroChina Company Limited  
Beijing 102206, China  
fanyugd@petrochina.  
com.cn

Yinliang Liu  
Production System Research  
Lab, CPPEI  
PetroChina Company Limited  
Beijing 102206, China  
liuyinliang@petrochina.  
com.cn

Hui Deng  
Production System Research  
Lab, CPPEI  
PetroChina Company Limited  
Beijing 102206, China  
denghui1984@petrochina.  
com.cn

Yang Lin  
Production System Research  
Lab, CPPEI  
PetroChina Company Limited  
Beijing 102206, China  
lin\_yang@petrochina.  
com.cn

## ABSTRACT

With the rapid construction of long-distance oil and gas pipelines, big pipeline network was gradually formed in China and its surrounding areas, therefore the management of pipeline production becomes increasingly complicated and difficult.

In this paper, we propose a data model for pipeline production to support the planning, scheduling, distribution, metering, energy consumption, process technology, professional computing and the other business in pipeline production. We also present a query language that can be used for discovering resources of interest in pipeline production based on the descriptions attached to the resources.

Following our model and based on the actual business needs, we have implemented a pipeline production management system (PPMS). The system provides services for group company, professional companies, regional companies and their affiliated grassroots units, and overseas companies to improve the efficiency and effectiveness of pipeline production management as well as safeguard the accuracy and the completeness of data.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous; D.2.8 [Software Engineering]: Metrics—*complexity measures, performance measures*

## General Terms

Theory

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

## Keywords

Data Model, Information System, Oil and Gas Pipeline, Pipeline Production Management

## 1. INTRODUCTION

In recent years, with the rapid construction of long-distance oil and gas pipelines, the big pipeline network was gradually formed in China and its surrounding areas. As of March 2013, a group company owns about 50,000 kilometers long-distance pipelines, of which approximately 33,000 kilometers of natural gas pipelines, 10,000 kilometers crude oil pipelines and 6,700 kilometers refined oil pipelines. The pipelines formed a cross-country and cross-region oil and gas pipeline network. This makes the production management of oil and gas pipeline network become more and more difficult and complex.

Roughly speaking, pipeline production management system is an information system managing the storage and transportation business of long-distance oil and gas pipelines. In particular, PPMS focused on the pipeline production business including: planning management, scheduling and operation, transportation and distribution, professional computing, statistics and analytics etc. In general, a pipeline production management system will be used to:

- standardize the management of pipeline production and operation,
- safeguard the accuracy and the completeness of data,
- improve the efficiency and effectiveness of pipeline production management, as well as
- reduce the burden of grass-roots staffs.

In order to carry out the above mentioned tasks, a pipeline production management system should be able to handle the entities and relationships in pipeline production in an accurate and flexible way, amongst which descriptions and their attachments to resources play an important role.

Combined with years of relevant professional pipeline production business knowledge and the previous experience of building,

operating and managing the information systems for pipeline production, in this paper, we propose a data model for pipeline production defining in a clear and rigorous way all the resources and relationships required to carry out the tasks of a pipeline production management system. The model is established based on the deep understanding of oil and gas pipeline production business and the related technologies, including the planning management, scheduling and operation, transportation and distribution, professional computing, statistics and analytics, Supervisory Control and Data Acquisition (SCADA) system, ERP system and etc. We formalize the model using first order logic, and the model places descriptions as first class citizens. We also present a query language that can be used for discovering resources in pipeline production based on the descriptions of resources.

The rest of the paper is structured as follows: Section 2 introduces the basic concepts of pipeline production, then Section 3 defines the formal data model for pipeline production along with an example, and Section 4 presents the query language. Section 5 discusses the implementation of a pipeline production management system. Section 6 briefly reviews the relevant work. Finally, Section 7 concludes the paper and outlines future work.

## 2. THE BASIC CONCEPTS OF PIPELINE PRODUCTION

Intuitively, we think of a resource as anything can be identified in pipeline production, such as a pipeline, a station, a place and so on. For the purposes of our discussions, we assume the existence of a set consisting of all resources that one can ever define in pipeline production.

We view each resource is associated with a content and a set of descriptions.

### 2.1 Content

In this work, by content of a resource  $r$  we mean the set of other resources that make up  $r$ ; each such resource is called a part of  $r$ . For example, each pipe segment (seen as a resource in its own right) can be seen as a part of the pipeline. Similarly, each station can be seen as a part of an organization.

#### 2.1.1 Concepts

To understand pipeline production business, basic knowledge of pipeline production is necessary, therefore the definitions of some important concepts in pipeline production are given as follows:

- Station, is the industrial place supporting the operation, maintenance, and monitoring of pipelines. Some of the stations also support the sales of gas or oil and the other pipeline production business. Along the pipelines, the stations provide different functions, for example, there are pump station, compressor station, metering station and so on. The valve chamber, gas storage, and oil depots along the pipelines are also classified into station. Station is the key node along the pipelines.
- Pipe Segment, is the pipe between two stations. Pipe segment is represented by a directed line segment.
- Pipeline, is the pipe connected by a plurality of directed pipe segments.
- Equipment, is the device used in pipeline production, such as Compressor, Chromatographic Analyzer, Flow Meter, Pump, Air Cooler, and other oil and gas pipeline monitoring and control devices.

- Organization, is the oil and gas pipeline production business-related company and the administrative department affiliated to the company.
- Customer, is the buyer of Natural Gas, Crude Oil, Compressed Natural Gas (CNG), or Liquefied Natural Gas (LNG).
- Supplier, is the company of supplying Natural Gas, Crude Oil, Compressed Natural Gas, or Liquefied Natural Gas.
- Transport Location, is the location where the gas or oil was transported away by train, boat, lorry or the other carriers except by the pipelines.

Here, we just list a few import concepts of our understanding in pipeline production, note that, the concepts may be defined differently by the other pipeline production business related organizations.

#### 2.1.2 Relations

In pipeline production, some of the resources are closely related, for instance, stations are connected by pipeline, there are multiple suppliers providing gas or oil for a station and etc. In order to meet the actual business needs, the relations in pipeline production should be described clearly and rigorously. Some of the relations in our pipeline production business are listed as follows:

- Pipe Segment and Station: each pipe segment is defined by the starting station and the ending station.
- Pipeline and Pipe Segment: pipeline is composed by a plurality of pipe segments in a certain order.
- Station and Organization: an organization can manage multiple stations.
- Pipeline and Organization: a pipeline can be managed by multiple organizations.
- Organization, Customer and Supplier: an organization can be a customer and a supplier at the same time.
- Equipment, Station and Organization: an equipment can be associated with multiple stations or multiple pipelines. By default, an equipment belongs to the organization that manages the station where the equipment is located at; if an equipment is associated with multiple stations, then the equipment belongs to the least upper bound organization that can manage all the stations that the equipment is associated with.
- Supplier, Pipeline, Station, and Transportation Location: a supplier could transport its oil/gas to a station (suppose identified by  $st$ ) by a pipeline; if in the station  $st$ , the oil/gas was transported to another place for distribution by train or boat or the other carriers except by pipeline, then the position of the station  $st$  is a transport location.
- Turnover between Pipelines: the transported oil/gas could be transferred from one pipeline to another pipeline through a station.

## 2.2 Description

A description of a resource  $r$  is the descriptive information of the resource. For example, the description of a station will include the location and the type of the station, while the description of a pipeline will include the length and the regions that the pipeline goes through and etc. A pipeline, however, can be described from

different points of view, each leading to a different description. As a result, a pipeline might be associated to a set of descriptions (and the same goes for a station). In order to accommodate several descriptions for the same resource, we will treat descriptions as resources in their own right.

### 2.2.1 Data Points

Data points are the important data that we should take special care of in pipeline production, so that to:

- monitor the operating status of pipelines (*e.g.* through the unit operation parameters);
- meter the transportation or distribution volume of oil/gas (*e.g.* through the flow-meter operation data);
- analyze the gas quality (*e.g.* through the chromatographic analyzer operation data);
- support the professional computing (*e.g.* calculating pipe deposit according to ISO 12213-2:2006 “Natural gas – Calculation of compression factor – Part 2: Calculation using molar composition analysis” [13]);
- generate different kinds of report forms (*e.g.* Energy Consumption Monthly Report of Natural Gas Station);
- provide decision support by statistics and analytics; and etc.

In pipeline production, to use data points effectively, data points must be associated with descriptions. For example, the rotation speed of compressor is a very important operating parameter used for monitoring the status of compressor; however, if we only have the rotation speed of a compressor (suppose marked by a data item *RotationSpeed*), then the *RotationSpeed* data will be useless, because to know the corresponding compressor, we need to find the identifier of the compressor, the location of the compressor and the other information related to the compressor. Therefore, descriptions should be attached to data points. For the above example, the data point *RotationSpeed* could be associated with *CompressorSN*, *Station*, *Time*, *Date* and the other related properties by descriptions.

### 2.2.2 Description Reuse

In pipeline production, descriptions should be reusable so that to reduce the workloads of describing similar resources. Generally, we have two ways of reuse:

- reuse without any customizations, which means a description will be reused directly without any modifications. There are thousands and tens of thousands of data acquisition points along the pipelines, some of the data points are the same type, therefore it is not a good idea if we need to create each time a similar description for each data point of the same type. For example, a station could have multiple compressors, the data acquisition points of compressor rotation speed are the same type, these data points could be described by reusing a same description.
- reuse with customizations, which means a description will be reused by extending the description with new properties. For example, to further describe the rotation speed of a compressor *RotationSpeed*, we could attach new properties for instance *InletPressure* and *OutletTemperature* to the description for *RotationSpeed*.

We note that descriptions will be defined according to schemas, for our purposes, we view a schema as consisting of a set of classes and/or a set of properties; additionally, a property could have one or multiple ranges and one or multiple domains. Our definition of classes and properties are similar to the ones in object-oriented modeling, in Description Logics [5], and in semantic web framework [6]. When defining a description, properties may come from one or more schemas.

## 3. PIPELINE PRODUCTION DEFINITION

### 3.1 The Language $\mathcal{L}$ of Our Model

The language that we propose for our data model is a function-free first-order language, the predicate symbols are listed as follows:

- $\text{Class}(s, c)$ , expresses the fact that  $c$  is a class defined in schema  $s$ .
- $\text{Property}(s, p)$ , expresses the fact that  $p$  is a property defined in schema  $s$ .
- $\text{Domain}(s, p, c)$ , expresses the fact that in schema  $s$  class  $c$  is the domain of property  $p$ .
- $\text{Range}(s, p, c)$ , expresses the fact that in schema  $s$  class  $c$  is the range of property  $p$ .
- $\text{IsaCl}(s, c_1, c_2)$ , expresses the fact that in schema  $s$  class  $c_1$  is a sub-class of class  $c_2$ .
- $\text{IsaPr}(s, p_1, p_2)$ , expresses the fact that in schema  $s$  property  $p_1$  is a sub-property of property  $p_2$ .
- $\text{Description}(d, s, p)$ , expresses the fact that property  $p$  over schema  $s$  belongs to description  $d$ .
- $\text{PrVal}(i, s, p, j)$  expresses the fact that  $i$  has a resource identified by  $j$  as value of property  $p$  from schema  $s$ .
- $\text{Cllnst}(i, s, c)$  expresses the fact that  $i$  is an instance of class  $c$  from schema  $s$ .
- $\text{PartOf}(i_1, i_2)$ , expresses the fact that  $i_1$  identifies a resource which is part of the resource identified by  $i_2$ .
- $\text{DescOf}(d, i)$ , expresses the fact that  $d$  is a description of  $i$ .

We denote the the above predicate symbols defined in first-order logic as  $\mathcal{L}$ . Before we proceed further with the definition of our data model for pipeline production, let’s see how a real example in pipeline production can be represented using  $\mathcal{L}$ .

### 3.2 An Example Using $\mathcal{L}$

Consider a SCADA data point “Compressor Rotation Speed”, we refer to it by an identifier *crs*. In our formulas, we represent strings by enclosing them between quotes, and resources as the values themselves italicized. Based on these conventions, we have the following formula:

$$\text{Cllnst}(\textit{crs}, s, \textit{CompressorRotationSpeed})$$

By the above formula, we understand that *crs* is an instance of class *CompressorRotationSpeed* and *CompressorRotationSpeed* is a class defined in schema  $s$ :

$$\text{Class}(s, \textit{CompressorRotationSpeed})$$

To describe the data point  $crs$ , we define a description  $d$  including multiple properties from schema  $s_1$  and  $s_2$ , and description  $d$  is represented as follows:

$Description(d, s_1, Name)$   
 $Description(d, s_1, CompressorSN)$   
 $Description(d, s_1, Speed)$   
 $Description(d, s_1, Time)$   
 $Description(d, s_1, Date)$   
 $Description(d, s_1, Station)$   
 $Description(d, s_2, InletPressure)$   
 $Description(d, s_2, OutletPressure)$

We understand in the above formulas,  $Name$ ,  $CompressorSN$ ,  $Speed$ ,  $Time$ ,  $Date$ , and  $Station$  are properties defined in schema  $s_1$ , while the properties  $InletPressure$  and  $OutletPressure$  are defined by schema  $s_2$ . They are represented as:

$Property(s_1, Name)$   
 $Property(s_1, CompressorSN)$   
 $Property(s_1, Speed)$   
 $Property(s_1, Time)$   
 $Property(s_1, Date)$   
 $Property(s_1, Station)$   
 $Property(s_2, InletPressure)$   
 $Property(s_2, OutletPressure)$

We now could assert that  $d$  is a description of  $crs$  by a formula:

$DescOf(d, crs)$

Finally, we could attach values to the properties of  $crs$  as follows:

$PrVal(crs, s_1, Name, \text{"Compressor Rotation Speed"})$   
 $PrVal(crs, s_1, CompressorSN, \text{"UZS-001"})$   
 $PrVal(crs, s_1, Speed, \text{"8 000"})$   
 $PrVal(crs, s_1, Time, \text{"13:12:11"})$   
 $PrVal(crs, s_1, Date, \text{"2013-11-01"})$   
 $PrVal(crs, s_1, Station, \text{"UZS"})$   
 $PrVal(crs, s_2, InletPressure, \text{"5 000 000"})$   
 $PrVal(crs, s_2, OutletPressure, \text{"8 000 000"})$

Note that, the above description  $d$  could be reused to describe other SCADA data points. For example, a compressor rotation speed  $crs_2$  of another compressor, which is the same type of  $crs$ , can be described by  $d$  as follows:

$DescOf(d, crs_2)$

We then can attach different values to the properties of  $crs_2$  using  $PrVal$  as follows:

$PrVal(crs_2, s_1, Name, \text{"Compressor Rotation Speed"})$   
 $PrVal(crs_2, s_1, CompressorSN, \text{"UZS-002"})$   
 $PrVal(crs_2, s_1, Speed, \text{"9 000"})$   
 $PrVal(crs_2, s_1, Time, \text{"23:12:11"})$   
 $PrVal(crs_2, s_1, Date, \text{"2013-12-01"})$   
 $PrVal(crs_2, s_1, Station, \text{"UZS"})$   
 $PrVal(crs_2, s_2, InletPressure, \text{"4 500 000"})$   
 $PrVal(crs_2, s_2, OutletPressure, \text{"7 000 000"})$

### 3.3 The Axioms $\mathcal{A}$ of Our Model

We now present the axioms of our model. The axioms of our model will be used to capture the meaning of the predicate symbols, as well as capture the implicit knowledge in pipeline production. Variables in the axioms are universally quantified.

(A1) If property  $p$  has class  $c$  as its domains in a schema  $s$ , then  $p$  and  $c$  must be defined in  $s$ :

$$\text{Domain}(s, p, c) \rightarrow (\text{Property}(s, p) \wedge \text{Class}(s, c))$$

(A2) If a property  $p$  has class  $c$  as its ranges in a schema  $s$ , then  $p$  and  $c$  must be defined in  $s$ :

$$\text{Range}(s, p, c) \rightarrow (\text{Property}(s, p) \wedge \text{Class}(s, c))$$

(A3) If  $c_1$  is a sub-class of  $c_2$  in a schema  $s$ , then  $c_1$  and  $c_2$  must be defined in  $s$ :

$$\text{IsaCl}(s, c_1, c_2) \rightarrow (\text{Class}(s, c_1) \wedge \text{Class}(s, c_2))$$

(A4) If  $p_1$  is a sub-property of  $p_2$  in a schema  $s$ , then  $p_1$  and  $p_2$  must be defined in  $s$ :

$$\text{IsaPr}(s, p_1, p_2) \rightarrow (\text{Property}(s, p_1) \wedge \text{Property}(s, p_2))$$

(A5) If  $i$  is an instance of class  $c$  over schema  $s$ , then  $c$  must be defined in schema  $s$ :

$$\text{CInst}(i, s, c) \rightarrow \text{Class}(s, c)$$

(A6) If a description  $d$  contains a property  $p$  over schema  $s$ , then  $p$  must be defined in schema  $s$ :

$$\text{Description}(d, s, p) \rightarrow \text{Property}(s, p)$$

(A7) Sub-class is reflexive:

$$\text{Class}(s, c) \rightarrow \text{IsaCl}(s, c, c)$$

(A8) Sub-class is transitive:

$$(\text{IsaCl}(s, c_1, c_2) \wedge \text{IsaCl}(s, c_2, c_3)) \rightarrow \text{IsaCl}(s, c_1, c_3)$$

(A9) Sub-property is reflexive:

$$\text{Property}(s, p) \rightarrow \text{IsaPr}(s, p, p)$$

(A10) Sub-property is transitive:

$$(\text{IsaPr}(s, p_1, p_2) \wedge \text{IsaPr}(s, p_2, p_3)) \rightarrow \text{IsaPr}(s, p_1, p_3)$$

(A11) If  $i$  is an instance of class  $c_1$  from schema  $s$ , and  $c_1$  is a sub-class of  $c_2$  in  $s$ , then  $i$  is also an instance of class  $c_2$  from schema  $s$ :

$$(\text{CInst}(i, s, c_1) \wedge \text{IsaCl}(s, c_1, c_2)) \rightarrow \text{CInst}(i, s, c_2)$$

(A12) If  $p_1$  is a sub-property of  $p_2$  in  $s$ , and  $j$  is a  $p_1$ -value of  $i$ , then  $j$  is also a  $p_2$ -value of  $i$ :

$$(\text{IsaPr}(s, p_1, p_2) \wedge \text{PrVal}(i, s, p_1, j)) \rightarrow \text{PrVal}(i, s, p_2, j)$$

(A13) If  $d$  describes  $i$  that is part of  $j$ , then  $d$  describes  $j$  too:

$$(\text{DescOf}(d, i) \wedge \text{PartOf}(i, j)) \rightarrow \text{DescOf}(d, j)$$

This axiom transfers descriptions from parts to the whole.

We denote the above set of axioms as  $\mathcal{A}$ , and we denote the theory defined by  $\mathcal{L}$  and  $\mathcal{A}$  as  $\mathcal{T}$ .

In pipeline production, an interpretation  $I$  is created by the contents and the descriptions manually inserted by users when they record information about pipeline production or gathered from the relevant systems such as SCADA system, ERP system etc. The resulting pipeline production is then given by applying the axioms to these facts. In order to make this concept more precise, we re-write the axioms  $\mathcal{A}$  of our data model for pipeline production in the form of a positive datalog program  $D_{\mathcal{A}}$  as follows:

Property( $s, p$ )	:-	Domain( $s, p, c$ )
Class( $s, c$ )	:-	Domain( $s, p, c$ )
Property( $s, p$ )	:-	Range( $s, p, c$ )
Class( $s, c$ )	:-	Range( $s, p, c$ )
Class( $s, c_1$ )	:-	IsaCl( $s, c_1, c_2$ )
Class( $s, c_2$ )	:-	IsaCl( $s, c_1, c_2$ )
Property( $s, p_1$ )	:-	IsaPr( $s, p_1, p_2$ )
Property( $s, p_2$ )	:-	IsaPr( $s, p_1, p_2$ )
Class( $s, c$ )	:-	ClInst( $i, s, c$ )
Property( $s, p$ )	:-	Description( $d, s, p$ )
IsaCl( $s, c, c$ )	:-	Class( $s, c$ )
IsaCl( $s, c_1, c_3$ )	:-	IsaCl( $s, c_1, c_2$ ), IsaCl( $s, c_2, c_3$ )
IsaPr( $s, p, p$ )	:-	Property( $s, p$ )
IsaPr( $s, p_1, p_3$ )	:-	IsaPr( $s, p_1, p_2$ ), IsaPr( $s, p_2, p_3$ )
ClInst( $i, s, c_2$ )	:-	ClInst( $i, s, c_1$ ), IsaCl( $s, c_1, c_2$ )
PrVal( $i, s, p_2, j$ )	:-	IsaPr( $s, p_1, p_2$ ), PrVal( $i, s, p_1, j$ )
DescOf( $d, j$ )	:-	DescOf( $d, i$ ), PartOf( $i, j$ )

Given an interpretation  $I$  which can be seen as a set of facts of pipeline production, then the above rules in  $D_{\mathcal{A}}$  will be applied in order to derive the minimal model of  $D_{\mathcal{A}}$  containing  $I$ . The minimal model will be a larger set of facts containing  $I$  as well as all the consequences of  $I$  according to  $D_{\mathcal{A}}$ . Based on the logical programming [7], the minimal model exists and is unique.

**DEFINITION 1. (Pipeline Production)** Let  $I$  be any interpretation of  $\mathcal{L}$ , we call pipeline production over  $I$ , denoted PP, the minimal model  $\mathcal{M}(D_{\mathcal{A}}, I)$  of  $\mathcal{A}$  that contains  $I$ .

## 4. QUERY THE DATA OF PIPELINE PRODUCTION

When searching the data of pipeline production, an intuitive and straightforward way of expressing the user's information need is to relate description to the sought resource.

For example, to search all the compressors in the station "UZS", the user could use a query like:

$$(\exists ?csn) \text{ PrVal}(?csn, s, \text{LocatedIn}, \text{"UZS"})$$

In the above query,  $?csn$  is a variable representing the compressor serial number,  $\text{LocatedIn}$  is a property defined in schema  $s$ .

For allowing queries to state simple conditions on property values, we consider any completion of PP endowed with six built-in relations, namely the  $=$ ,  $\neq$ ,  $>$ ,  $<$ ,  $\leq$  and  $\geq$  relations; we consider these relations as built-in predicates, therefore not subject to the completion of PP.

To exemplify, let us consider again the "Compressor Rotation Speed" example. When searching for the time when the compressor "UZS-001" was not running in the day of 2008-09-25, the user

could use a query  $Q_t$  like:

$$\begin{aligned} (\exists ?t) \quad & \text{PrVal}(?crs, s, \text{Time}, ?t) \wedge \\ & (\text{PrVal}(?crs, s, \text{Speed}, ?rs) \wedge (?rs < \text{"100"})) \wedge \\ & \text{PrVal}(?crs, s, \text{CompressorSN}, \text{"UZS-001"}) \wedge \\ & \text{PrVal}(?crs, s, \text{Date}, \text{"2008-09-25"}) \end{aligned}$$

In the above query,  $?t$ ,  $?crs$  and  $?rs$  are variables occur in  $Q_t$ , and  $?t$  represents the time,  $?crs$  represents the identifier of compressor rotation speed,  $?rs$  represents the compressor rotation speed. Note that, the compressor rotation speed can be used to judge the running status of a compressor, *i.e.* if the rotation speed was less than 100, we consider the compressor was not running.

**DEFINITION 2. (Query over Pipeline Production)** A query over pipeline production PP is well-formed formula  $Q(x_1, \dots, x_n)$  of  $\mathcal{L}$ , in which  $x_1, \dots, x_n$  are free variables and  $n \geq 1$ .

The answer of a query with  $n$  ( $n \geq 1$ ) free variables is the set of resources  $\langle r_1, \dots, r_n \rangle$  such that, when every variable  $x_i$  is bound to the corresponding resource  $r_i$ , the resulting formula of  $\mathcal{L}(r_1, \dots, r_n)$  is true in PP. Formally, we have the following definition for the answer of a query.

**DEFINITION 3. (Answer of A Query)** The answer of a query  $Q(x_1, \dots, x_n)$  over pipeline production PP is given by:

$$\text{answer}(Q, \text{PP}) = \{ \langle r_1, \dots, r_n \rangle \mid Q(r_1, \dots, r_n) \in \text{PP} \}$$

## 5. PIPELINE PRODUCTION SYSTEM

We have adopted first-order logic for modeling pipeline production. The choice of logic was deliberate in order to be able to describe the pipeline production without being constrained by any technical considerations. However, the goal of our work is to contribute to the *management system* of pipeline production. Therefore, we now consider how our model can be implemented. We consider two different scenarios:

- The first scenario consists in implementing the model with relational database and computing the completion of pipeline production via a datalog engine [8, 9, 10, 11, 12]. This implementation is conceptually straightforward, as an interpretation of  $\mathcal{L}$  consists just of a set of relations that can be implemented as tables.
- The second scenario consists in implementing the model with Resource Description Framework (RDF) [1], and using an RDF inference engine for computing the completion of the pipeline production. This implementation is conceptually much more complicated, as it requires translating the relations and the axioms of the model in RDF.

The first scenario exploits the well-established relational technology, including the optimized query processing of SQL. This guarantees scalability and robustness of the implementation. The second scenario benefits from the fact that RDF is a generally accepted representation language in the context of the Semantic Web. Although RDF has not yet achieved the maturity of relational technology, tools for managing RDF graphs have been intensely researched and developed in the last decade, and are now reaching a significant level of technological maturity. Such tools include systems for the persistence of large RDF graphs, for instance [3], RDF inference engines and optimized query processing engines for SPARQL [2], *e.g.* [4].

## 5.1 Relational Implementation

By choosing an implementation strategy based on relational technologies, we can benefit at a minimal effort of the scalability and the optimized query evaluation of relational database management systems (RDBMSs).

A simple strategy for implementing the model could then consist of the following steps:

- Store the initial interpretation  $I$  of pipeline production to a relational database  $RDB(I)$ ; the mapping from  $I$  to  $RDB(I)$  is straightforward.
- Compute the completion of  $RDB(I)$  to obtain the database  $RDB(\mathcal{M}(D_{\mathcal{A}}, I))$  via a datalog engine [8, 9, 10, 11, 12]; this requires adding tuples to the tables in  $RDB(I)$  using the inference mechanism that we have described in  $\mathcal{A}$ .
- Map each query  $Q$  against the pipeline production to an equivalent SQL query  $SQL(Q)$ .

To implement our data model for pipeline production by relational database, we should pay attention to the design of algorithms for maintaining  $RDB(\mathcal{M}(D_{\mathcal{A}}, I))$  in the condition of user updates. We also should take care in choosing a suitable datalog engine.

## 5.2 RDF based Implementation

RDF is a knowledge representation language for describing resources using triples of the form (subject, predicate, object). In a triple, the subject can be a URI or a local identifier (also called blank node) for unnamed entities; the predicate can only be a URI; the object can be a URI, a local identifier or a literal (*i.e.* a string of characters). The predicate in an RDF triple specifies how the subject and the object of the triple are related.

A set of RDF triples is called an RDF graph. This graph is obtained by interpreting each triple as a labelled arrow having the subject as its source, the object as its target and the predicate as its label. RDF has a formal semantics on top of which an inference mechanism is defined. This mechanism allows expanding a given RDF graph by adding to it the new triples that can be inferred from existing ones.

In order to implement our model in RDF, we must provide means for:

- mapping the relations of  $\mathcal{L}$  into equivalent RDF graphs,
- mapping the inference mechanism of our data model into that of RDF, and
- translating the query language of our model to SPARQL.

## 5.3 System Development

As we have already observed, implementing our model using RDF is more complicated, while implementing our model using relational technologies is straightforward. For this reason and due to the time limit for the project of pipeline production management system, we implemented the pipeline production data model using relational technologies.

Based on the investigation report and solution design, pipeline production management system should provide the following business functions for the users:

- Planning Management, manages yearly agreement intake and transportation volume of gas/oil, yearly planned production volume of gas/oil, subdivided monthly production plan, and etc.

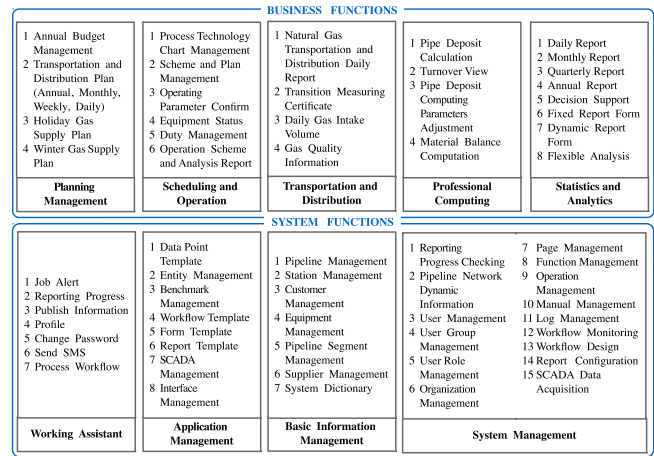


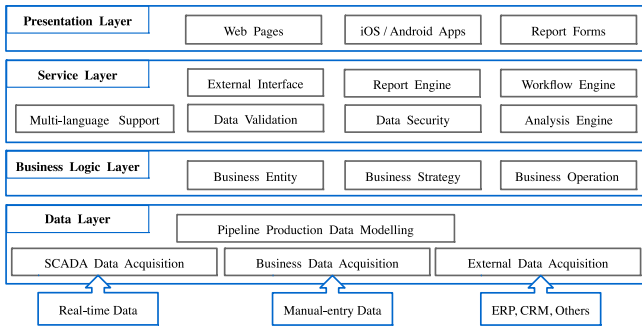
Figure 1: Function Architecture of Pipeline Production Management System

- Scheduling and Operation, manages station daily pipeline production operating data, production logs, unit operating data, flow-meter operating data, chromatographic analyzer operating data, as well as generates production operating daily report and hourly parameter summary table.
- Transportation and Distribution, manages the daily gas/oil intake volume, daily gas/oil distribution volume, gas/oil quality information and etc.
- Professional Computing, provides pipeline production professional computing services. For instance, pipe deposit calculation according to the ISO 12213-2:2006 “Natural gas – Calculation of compression factor – Part 2: Calculation using molar-composition analysis” [13].
- Statistics and Analytics, generates different kinds of reports including production daily report, production monthly report, operation daily report, energy consumption report, and so on for the group company headquarters, the regional companies, and the vendors, as well as provides decision support.

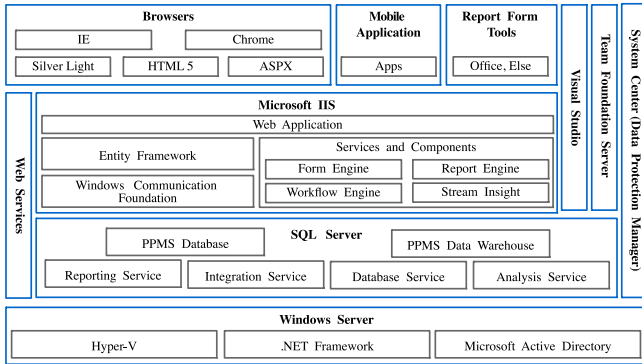
The pipeline production business functions are supported by the following system functions:

- System Management, configures the system functions, such as the user group creation based on users’ authority layers, user role configuration based on user’s duties, work-flow management based on business process, and etc.
- Basic Information Management, manages the basic information of the system such as pipeline basic information maintenance, pipeline network and station visualization, pipeline running dynamic visualization, supplier and customer information management, and etc.
- Application Management, manages the configurable applications such as the configuration of different form templates for different stations, configuration of different reports based on different companies’ requirement, and etc.
- Working Assistant, enhances production management efficiency by the functions of job alerts, work-flow process, system information publish, short message service, personal information management, and the other auxiliary functions.





**Figure 2: System Architecture of Pipeline Production Management System**



**Figure 3: Software Architecture of Pipeline Production Management System**

The function architecture of the pipeline production management system is summarized in Figure 1.

The data of PPMS comes from SCADA system, manual entry, and the external systems (ERP, CRM and etc) of the enterprise, the system architecture of pipeline production management system is presented in Figure 2.

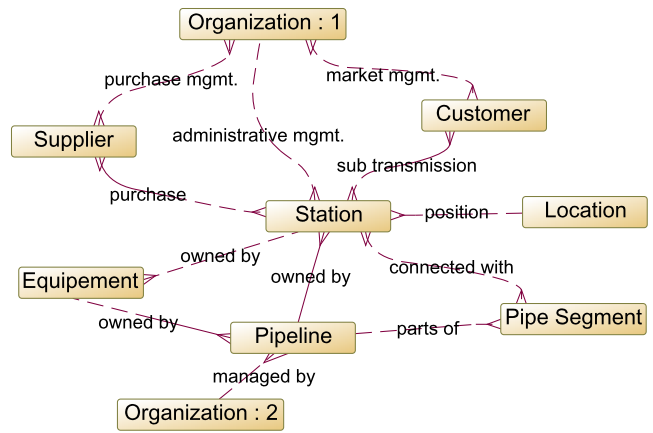
PPMS was developed with .NET and the related softwares and technologies, the software architecture of pipeline production management system is given by Figure 3.

The hardwares for supporting the running of PPMS are listed in Table I.

A simplified ER diagram of PPMS is demonstrated in Figure 4.

A few ways were used to ensure the security of pipeline production management system:

- communications security: strict access control was provided by a firewall that isolates the enterprise internal network from the outside world; Demilitarized Zone (DMZ) and HTTPS encryption were used to protected external access to PPMS; in addition, hardware encryption (USB key) was used for PPMS user authentication.
- data security: local and off-site data backups were performed regularly (in seconds).
- system application security: hot standby mechanism was established for important services; virtualization technology



**Figure 4: A Simplified ER Diagram of Pipeline Production Management System**

**Table I: The Hardwares for Running PPMS**

Device	Device Requirements and Configuration	Qty
Web Server	10 core 2.26GHz Intel Xeon E7 series processor × 4, 256GB DDR3 memory, 8GB HBA card × 2, 600GB SAS hot plug hard drive × 5, Hyper-V supported	6
Report Form Server	10 core 2.26GHz Intel Xeon E7 series processor × 4, 64GB DDR3 memory, 600GB SAS hot plug hard drive × 5, Hyper-V supported, dual backup required	2
Operation Monitoring Server	ditto, no dual backup required	2
Application Server	ditto, dual backup required	3
Data Acquisition Server	ditto	2
Front End Server	8 core 2GHz Intel Xeon E5 series processor × 2, 32GB DDR3 memory, 600GB SAS hot plug hard drive × 5	1
Database Server	10 core 2.26GHz Intel Xeon E7 series processor × 4, 256GB DDR3 memory, 8GB HBA card × 2, 600GB SAS hot plug hard drive × 5, Hyper-V supported, double cluster deployment	4
ESI Load Forecasting Server	10 core 2.26GHz Intel Xeon E7 series processor × 4, 64GB DDR3 memory, 600GB SAS hot plug hard drive × 5, Hyper-V supported	1
Backup Server	8 core 2GHz Intel Xeon E5 series processor × 2, 32GB DDR3 memory, 600GB SAS hot plug hard drive × 5	1
Storage Device	Dual controller, buffer capacity of 24G, 16 front-end 8Gb fiber ports, 8 back-end 6Gb ports, 4 Gigabit NAS network ports; SAN architecture drive bare capacity 30TB, NAS architecture drive bare capacity 20TB; 24-port optical switch, configure and activate 16 ports	1
Load Balancer	12 Gigabit ports, 4 core processor, 4Gbps throughput, the maximum number of concurrent connections: 8000000, 8GB DDR3 memory, double storage medium	2
Ethernet Switch	48-port Gigabit switch, redundant power supply	2
KVM	16 ports, with remote management function	1
Disk Array	Logical backup capacity more than 100TB, 2 B/Gb FC ports, 2 Gigabit ethernet ports	1

was used to enhance the high availability of web servers.

The pipeline production management system has been launched for a few months, providing around 276 professional services for managing the group company's pipeline production business within China. The number of ingested data is around 622800 items per day. The system was also extended to manage the pipeline production of Turkmenistan, Uzbekistan, Kazakhstan and China gas pipelines and the pipeline production of Sino-Burma gas pipelines.

## 6. RELATED WORKS

To the best of our knowledge, our model is the first data model specializing in the production management of pipelines. Although there are some models for pipelines such as Pipeline Open Data Standard (PODS) [14], ArcGIS Pipeline Data Model (APDM) [16], and PODS ESRI Spatial [15], these models focus on the management of pipeline integrity, *i.e.* integrated with Geographic Information System (GIS) information, *i.e.* these models help pipeline operators to collect, verify, analyze, and maintain the information

about physical segment of pipeline, so that to support the re-route, change of service, abandonment, removal, repair, and replacement of pipeline [14].

Entity-Relationship (ER) model [17] is a data model for describing data in an abstract and conceptual way, the essential elements of ER model are: Entity Set, Relationship Set, and Attribute. Entity is an object that can be uniquely and distinctly identified, Entity Set is a set of entities of the same type that has the common properties. Relationship is the association among entities, Relationship Set is a mathematical relationship among  $n \geq 2$  entities. Attribute is the descriptive information about an entity or a relationship. The ER model describes only the static view of data. Unified Modeling Language (UML) [18] is a widely accepted object-oriented modeling language that has many components to model different aspects of an entire system graphically; Class Diagram is the closest component of UML corresponds to ER Diagram, but several differences [19]. In our model, we use relations to express the basic facts in pipeline production, and rely on a first-order theory to derive information implicit in the given facts, under certain axioms; descriptions are placed as first class citizens and are defined independently of the resources they might be associated with, this is different from the systems often used to implement ER models, which generally assume that attributes of an object are stored in the object; in addition, our model supports the reuse of descriptions. The choice of logic for modeling is motivated by the desire of generality, which includes freedom from any technological constraint.

Enterprise Resource Planning (ERP) is a system focused on business management, to improve the efficiency and effectiveness of business processes so that to reach the business goals. Customer Relationship Management (CRM) is a system for a company to manage current and future customer interactions, to improve customer relations so that to increase brand loyalty and profits. However, neither ERP nor CRM can satisfy the requirements of our pipeline production management. In fact, part of the customer information from CRM and some of the ERP modules such as PM (Plant Maintenance), MM (Materials Management) and PS (Project Systems) are the data sources of pipeline production management system, in turn, PPMS supports the FI (Financial Accounting) module of ERP.

## 7. CONCLUSIONS

In this paper, we have defined a data model for pipeline production based on two basic concepts: content and description. The two concepts are expressed by a certain set of predicates using a first-order logical approach, the axioms of the model are defined to fix the semantics of the predicates and to capture the implicit knowledge in pipeline production. In our model, descriptions are placed as first class citizens with their own identifiers and are defined independently of the resources they might be attached to, additionally descriptions are flexible for extension and reuse. We also present a query language for discovering resources of interest in pipeline production based on the descriptions attached to the resources.

Our model provides a convenient and flexible way for describing the concepts, the properties, and the relations in pipeline production. The model also leads to an efficient communication between the business people defining the pipeline production management requirements for an information system and the technical people developing the information system in response to those requirements.

Following our model, we implemented a pipeline production management system for managing the group company's pipeline production business in China as well in some overseas regions. The pipeline production management system received very high ap-

praisal from the users of the group company, the regional companies, and the overseas companies.

Regarding further work, we will continue improving our model. Furthermore, there is one direction which is promising in our opinion that the group company has many different information systems, which are categorized into ERP Systems, Petroleum Exploration and Development Systems, Refining and Marketing Systems, Service and Support Systems, Infrastructure and Security Systems, and so on. We are currently working towards the extension of our model to manage the big data across these above mentioned systems to allow all information systems of the enterprise to operate together in a cooperative manner, so that to maximize the overall data management and analysis benefit to the enterprise.

## 8. REFERENCES

- [1] Resource Description Framework (RDF). <http://www.w3.org/RDF/>
- [2] SPARQL 1.1 Overview, W3C Recommendation 21 March 2013. <http://www.w3.org/TR/sparql11-overview/>
- [3] Semantic Web Development Tools. <http://www.w3.org/2001/sw/wiki/Tools>
- [4] SPARQL Implementations. <http://esw.w3.org/SparqlImplementations>
- [5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi and Peter F. Patel-Schneider. The description logic handbook: theory, implementation, and applications. Cambridge University Press, 2003.
- [6] Frank Manola and Eric Miller (Editors). RDF Primer, W3C Recommendation 10 February 2004. <http://www.w3.org/TR/rdf-primer/>
- [7] J. W. Lloyd. Foundations of Logic Programming. Springer-Verlag New York, Inc., 1987.
- [8] Datalog. <http://www.ccs.neu.edu/home/ramsdell/tools/datalog/>
- [9] Clojure Datalog. <https://code.google.com/p/clojure-contrib/wiki/DatalogOverview>
- [10] IRIS Reasoner. <http://iris-reasoner.org/>
- [11] 4QL. <http://4ql.org/>
- [12] Datalog Educational System (DES). <http://www.fdi.ucm.es/profesor/fernand/DES/>
- [13] ISO 12213-2:2006 Natural gas – Calculation of compression factor – Part 2: Calculation using molar-composition analysis. [http://www.iso.org/iso/home/store/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=44411](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=44411)
- [14] Pipeline Open Data Standard. <http://www.pods.org/>
- [15] PODS ESRI Spatial. <http://www.pods.org/75/PODS%20ESRI%20Spatial%20Geodatabase/>
- [16] ArcGIS Pipeline Data Model (APDM). <http://www.esri.com/industries/pipeline/community/datamodel>
- [17] Peter Pin-shan Chen. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems, volume 1, pages 9–36, 1976.
- [18] Unified Modeling Language (UML). <http://www.omg.org/spec/UML/>
- [19] Bernadette Marie Byrne and Yasser Shahzad Qureshi. The Use of UML Class Diagrams to Teach Database Modelling and Database Design. Proceedings of the 11th International Workshop on the Teaching, Learning and Assessment of Databases, University of Sunderland, 5 July 2013.

# Renewable Energy Data Sources in the Semantic Web with OpenWatt

D. Davide Lamanna  
Sapienza Università di Roma  
Rome, Italy  
lamanna@dis.uniroma1.it

Antonio Maccioni  
Università Roma Tre  
Rome, Italy  
maccioni@dia.uniroma3.it

## ABSTRACT

Although the sector of renewable energies has gained a significant role, companies still encounter considerable barriers to scale up their business. This is partly due to the way data and information are (wrongly) managed. Often, data is: partially available, noisy, inconsistent, sparse in heterogeneous sources, unstructured, represented through non-standard and proprietary formats. As a result, energy planning tasks are semi-automatic or, in the worst cases, even manual. As a result, the process that uses such data is exceedingly complex and results to be error-prone and ineffective. OPENWATT aims at establishing an ideal scenario in the renewable energy sector where different categories of data are fully integrated and can synergically complement each other. In particular, OPENWATT overcomes existing drawbacks by introducing the paradigm of Linked Open Data to represent renewable energy data on the (Semantic) Web. With OPENWATT, data increases in quality, tools become interoperable with each other and the process gains in usability, productivity and efficiency. Moreover, OPENWATT enables and favours the development of new applications and services.

## Keywords

OpenWatt, Renewable Energy, Ontology, Linked Open Data, Web of Data

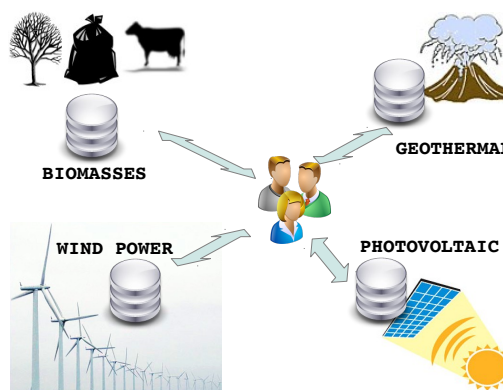
## 1. INTRODUCTION

Renewable Energy (RE) has a prominent market share. It will increase its popularity in the near future due to sustainability issues that are affecting the Earth.

**Motivation.** The report Energy [R]evolution [18] shows that employees in this sector are around the 50% of the overall energy sector. On the other hand, RE satisfies only 12.5% of European energy consumption. It sounds like the potential production and consumption are a mirage and the

Europe 20-20-20<sup>1</sup> target is far to be reached. Actually, companies in this sector are not able to scale up their business as it should be. We ascribe such lack to the wrong way data and information are managed. Nowadays, businesses can grow quickly only if information and knowledge management is efficient, e.g., Amazon, Google, Twitter.

When a player in this sector has to conduct an energy plan or has to take important decisions about a territory, she has a limited scope of available information, because they are closed in several data silos, they use different scale, they refer different locations, they use different schemas; not to mention the not-so-rare presence of inconsistencies and errors. Correlating and integrating automatically these data is unfeasible. It follows that, the planning process requires significant manual operations and final results have approximations that can incur into errors. These deficiencies affect the tools that support business decisions. Figure 1 shows the fragmentation of actual scenario of reference, where several data sources are separated and the integration consists of different processes conducted by humans.



**Figure 1: The actual scenario of renewable energy data management.**

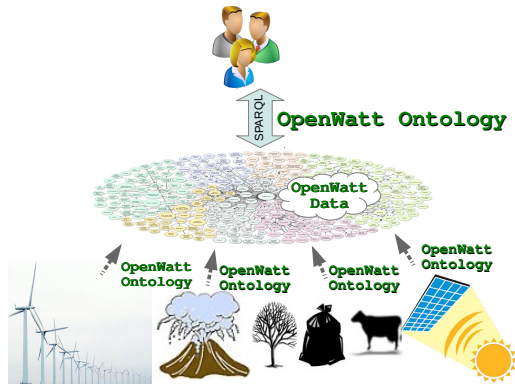
In addition, since the scope is limited and the uniformity is poor, the tools are vertical. It means that they pursue a technology-driven approach instead of a potentiality-driven approach. They do not fully consider the potentialities of available resources in a territory. For example, let us think of the agricultural contexts where usually only solar energy is considered in the planning, while other sources (e.g., biomasses, wind power) are abundantly available. Decision

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

<sup>1</sup> [http://ec.europa.eu/europe2020/index\\_en.htm](http://ec.europa.eu/europe2020/index_en.htm)

tools leave further big potential unexploited. Issues like these prevent the business in this sector to expand properly. OPENWATT aims at being the answer to such issues, thus providing a valid support for energy designers, local public administrations and private bodies to evaluate the energetic potential of a territory.

**Contribution.** OPENWATT<sup>2</sup> attempts to change the management of the data in the RE sector in order to overtake actual barriers. It creates a unique global database over a single schema for data about solar energy, wind energy, biomasses, etc. Data will be integrated, tools will be interoperable at data-level and, consequently, processes will be automatic. In this way, people can make rational and convenient choices and it will be much easier to guarantee self-sustainability of territories.



**Figure 2: The scenario of renewable energy data management with OpenWatt.**

To this aim, OPENWATT introduces the use of Semantic Web technologies and Linked Data for energy data management problems of modelling and integration, which are recently arising [14]. The benefits that Linked Data bring about in the renewable energy sector are evident to many [1] but still there is no adopted ontology nor Linked Data to be exploited. More in details, the contributions of OPENWATT are the following:

1. Definition of a common schema (i.e. an ontology) for renewable energy data;
2. Creation of a global instance of (Linked Open) renewable energy data;

The contribution in 1) allows to share the meaning of concepts and information. In this way, we can spread the (re-)use of OPENWATT data at global scale. The definition of the OPENWATT ontology is required to guarantee interoperability among data and applications. It facilitates the understanding by both humans and automatic agents. Consequently, it will be easier to compute automatic elaborations and inferences over the data. The contribution in 2) creates the OPENWATT “data common” of available data within the Web of Data. As we will deepen in the following of the paper, we started by considering the datasets that are actually used for planning purposes. Integrating data by tracing relationships through links makes value of individual data higher. This happens because a single information

<sup>2</sup> <http://openwatt.net/>

is enriched by the content of data connected to it. At the moment, the RE sector is a closed world of data, applications and human resources. Hence, the greater impact of OPENWATT is the breakdown of the state of the art in this sector in order to open it up towards a new scenario based on innovation, creativity and collaboration.

In fact, since OPENWATT data will be available on the Web for everyone, people will be enabled to develop applications and services. It is a fundamental drive to open the market and create business opportunities. From the software engineering point of view, the advantage is twofold: from one hand, they can develop applications without spending much time in understanding the meaning of data (nor permission to access them); on the other hand, applications can benefit from the fact that the application back-end is leaner, as it does not need to manage a persistence layer (i.e. a database) nor duplicate and include big datasets. This is particularly beneficial to mobile applications, which have limited resources and cannot embed large datasets or compute real-time data integrations. This scenario is summarized by Figure 2, where different data sources directly feed the Web of Data. On the other side, applications and human users use a single point of access (i.e. the Web of Data) by means of the W3C standards for the Semantic Web.

**Outline.** The rest of the paper is organized as follows. In Section 2 we introduce some preliminary notion as well as some related work. Section 3 overviews the project, focusing in particular on the methodology to generate the data and to design the ontology. The impact and the perspectives are analysed in Section 4. In Section 5, we discuss the development of the project. Finally, in Section 6, we draw some conclusions and sketch some future perspectives for OPENWATT.

## 2. BACKGROUND

This section introduces the technological and social scenario where OPENWATT fits.

**Open Data.** The social, economical and technological scenario (i.e. crowd-sourcing, Web 2.0, mobile apps, etc.) contributed to create the Open Data concept, that is the movement of individuals and organizations who believe that public data should not “live” enclosed within data silos, but they have to be freely available on the Web. As a result, organizations such as government bodies and public administrations (PAs) have started to publish online their data [10]. Obviously, the opening and publication of data in a “raw” format is not sufficient to guarantee high quality, interoperability and re-use in the development of applications. It is crucial to provide them in a standard and machine-readable form. These issues are addressed by the protocols of the Semantic Web stack and implemented by Linked Data as explained in the following. This is the reason why many Open Data initiatives follow the Linked Data principles [4].

**Semantic Web and Linked Data.** The vision of the Semantic Web is to introduce a global knowledge base where data is freely available on the Web and semantically organized through the so called ontologies of reference, described with RDF and “linked” to other data. This underlies the re-

placement of a Web of linked documents with a proper linked information space (Web of Data) where data are being enriched and inferred (Web of Meaning). These two concepts of Web of Data and Web of Meaning are often considered together as Semantic Web.

```
<rdf:RDF xmlns:owl =
  "http://www.w3.org/2002/07/owl#"
  xmlns:rdf =
  "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:skos =
  "http://www.w3.org/2004/02/skos/core#"
  xmlns:rdfs =
  "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:ow = "http://www.openwatt.net/">

<owl:Ontology rdf:about="http://openwatt.net/">

<owl:Class rdf:about="Municipality"></owl:Class>
<owl:Class rdf:about="Province"></owl:Class>

<owl:Class rdf:about="Region"></owl:Class>
<owl:Class rdf:about="Country"></owl:Class>
<owl:Class rdf:about="Province"></owl:Class>

<owl:Class rdf:about="Typology"></owl:Class>
<owl:Class rdf:about="Category"></owl:Class>
<owl:Class rdf:about="Measure"></owl:Class>

<owl:Class rdf:about="Potential">
  <rdfs:subClassOf rdf:resource="Typology"/>
</owl:Class>

<owl:Class rdf:about="Census">
  <rdfs:subClassOf rdf:resource="Measure"/>
</owl:Class>

<ow:Category rdf:about="Solar">
  <skos:narrower rdf:resource="Category" />
</ow:Category>

<ow:Category rdf:about="WoodyCrop">
  <skos:narrower rdf:resource="Biomass" />
</ow:Category>
...
<rdf:Property rdf:about="measured">
  <rdfs:domain rdf:resource="Typology" />
  <rdfs:range rdf:resource="Estimation" />
</rdf:Property>
</owl:Ontology>
</rdf:RDF>
```

Figure 3: The OpenWatt OWL Ontology.

To pursue this vision, Tim Berners-Lee proposed four principles to follow when opening data on the Web [4]: 1) use URIs to identify things; 2) use http URIs so people can look things up; 3) provide useful data in standard RDF; 4) use RDF to link to other things. Upon these principles, an Open Data rating system consisting of five levels indicated through the star symbol ★ was proposed<sup>3</sup>.

**Related Work.** Semantic Web technologies tackle data

<sup>3</sup> <http://5stardata.info/>

management tasks differently with respect to existing paradigms. For this reason, they have been proposed in different contexts as, for instance, to enable a Social Semantic Web [11]. In the energy data management, Semantic Web and Linked Data are starting to be taken into consideration [15, 8, 7, 16]. Authors of [15] and [8] use Linked Data to solve existing problems in smart environments and in domotics, including the energy management. EDF, the French electricity company uses semantic technologies to model its energy domain [7]. Linked Data principles inspired the design of an architecture for decentralised Smart Grid systems [16]. OEI (Ontology for Energy Investigations) [9], takes inspiration from the Ontology for Biomedical Investigations (OBI) to present a core ontology for energy systems. Authors of [5] state the intention to base on OWL a reasoner for their semantic modelling of life cycle assessment for energy environmental impact. The non-profit organisation for Renewable Energy and Energy Efficiency Partnership (REEP<sup>4</sup>) has the purpose to use renewable technologies to make improvements in the developing world. It includes 45 governments among its 385 partner organisations. REEP manages a Web portal<sup>5</sup> where Linked Data about partners, projects and available datasets are provided [3]. It can be possible that in the future OPENWATT will be part of the REEP data.

### 3. OPENWATT: DATA AND ONTOLOGY

To develop OPENWATT, we adopted a methodology that leads to the generation of the ontology of the data.

**Methodology.** The methodology is composed of the following steps:

1. *Data gathering:* we listed all the data available on the Web (through dumps of the datasets) that are useful for our aims, that are basically, the data actually used for the energy planning. For each dataset, we tracked all its descriptive information (e.g., meaning of the records, update frequency, licensing, etc.).
2. *Data cleaning:* this step regards the data quality. The data is analysed (in order to find inconsistencies, missings, etc.), cleaned (in order to correct errors) and normalized (in order to homogenize similar fields and different formats). At the end of the step, the datasets are more accurate and ready to be elaborated.
3. *Data modelling:* we developed a conceptual model comprising all the concepts needed and we defined relationships among these elements. We assigned names to the concepts and we defined an URI policy. Note that we considered the concepts that are represented in the datasets found in Step 1, but we did not take into consideration how they were originally modelled. Hence, we modelled them from scratch, it would have been impossible to apply a reverse engineering approach of different datasets.
4. *Data recognition:* we compared the data against the created model in order to find logical inconsistencies. At the same time, we checked if some of the concepts can be represented through existing ontologies. This

<sup>4</sup> <http://www.reeep.org/>

<sup>5</sup> [reegle.info](http://reegle.info)



increases interoperability and understandability of our data. In particular, we reused existing concepts for geographical locations and people (e.g., core public location vocabulary<sup>6</sup>).

5. *Ontology definition*: the final model is formalized using the W3C standard language for ontologies on the Web, that is OWL<sup>7</sup>. The output of this step will be explained better in the following.
6. *Data generation*: we transformed our data in RDF conforming the OPENWATT ontology. The output of this phase is explained better in the following.
7. *Metadata generation*: we enriched the information about the datasets by adding metadata, e.g., about the provenance of the datasets, the last modified date, the keywords describing the data, etc.
8. *External linking*: in this phase we linked our data to data already present in the Web of Data. External linking is intended as the process to produce RDF triples which subjects and objects belong to different datasets, that is, to bind related entities. Usually, these links represent the identity relationship between entities representing the same thing. The OWL property `sameAs` is used for the purpose. In our case, we linked our geographical entities to others already present on the Web (e.g., DBPedia<sup>8</sup>, Geonames<sup>9</sup>, etc.). This task is usually performed with record linkage tools. After this step, data is assessed with 5 stars.
9. *Data and ontology validation*: similarly to every other development process, a testing and validation task is performed. We have two different kinds of validation: syntactic and logical. The first checks if the syntax of both data and ontology are compliant to the W3C standards. Logical validation uses test cases that have to be satisfied. They are expressed through the definition of SPARQL queries. Logical problems may emerge and require the ontology to be modified, thus repeating the tasks from the third.
10. *Data publication*: we loaded our data on a triple store exposing a SPARQL endpoint<sup>10</sup>. The endpoint is not available publicly at the moment of the issue of this paper. Once it will be published on the Web, URIs will be dereferenceable [17].

Note that some of these steps have no strict precedence over others. So, the methodology can be repeated with a different order of the steps.

**Ontology.** This paragraph discusses the OPENWATT ontology. Figure 4 shows the main concepts of the ontology, while Figure 5 shows an excerpt of the ontology in RDF/XML format. They point out that other ontologies such as SKOS [12], PROV [13] and GeoNames<sup>11</sup> have been used in OPENWATT.

<sup>6</sup> [https://joinup.ec.europa.eu/asset/core\\_location/home](https://joinup.ec.europa.eu/asset/core_location/home)  
<sup>7</sup> Web Ontology Language - [www.w3.org/TR/owl-features/](http://www.w3.org/TR/owl-features/)  
<sup>8</sup> <http://dbpedia.org/>  
<sup>9</sup> <http://www.geonames.org/>  
<sup>10</sup> SPARQL Query Language for RDF - [www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)  
<sup>11</sup> <http://www.geonames.org/>

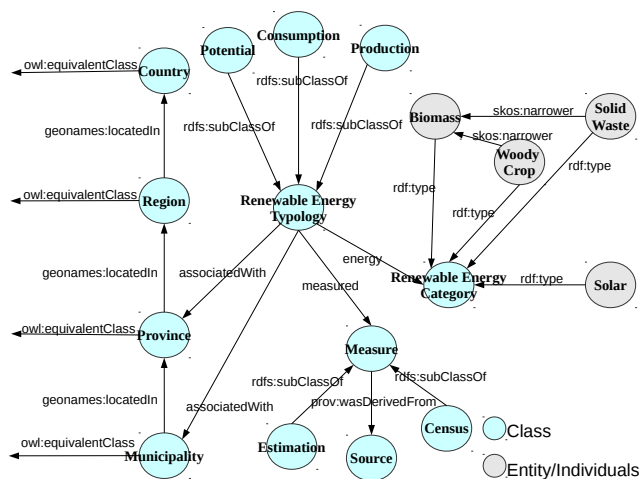


Figure 4: The OpenWatt ontology graph.

The core classes of the ontology are **Typology** (i.e. Renewable Energy Typology), **Category** (i.e. Renewable Energy Category) and **Measure**. We have three sub-classes of **Typology**: **Consumption**, **Potential** and **Production**. Every energy data value is instance of one of these classes. Their names are self-explanatory. They are also associated to (through the property `associatedWith`) a geographical location. These associations are mutually exclusively, in the sense that only one among the location taxonomy formed by **Country**, **Region**, **Province** and **Municipality** classes is valid. **Category** is a taxonomy of the existing renewable energy sources (e.g., **Biomass**, **Solar**, **Solid Waste**, etc.) implemented through SKOS [12]. **Measure** describes how the data was collected (e.g., **Estimation**, **Census**, etc.) and from which source (i.e. **Source**). Some of the relationships (i.e. `energy`) represented in Figure 4 denotes that the incident class nodes are domain and range of that property.

**Data.** OPENWATT contains the data we have gathered, but modelled over the ontology of the previous paragraph. In Figure 5 we have instantiated and described sample entities to understand better how data is modelled, which the full Turtle<sup>12</sup> serialization is in Figure 6. In the example, we have the entity `ent1` representing a potential of 10.4 GW coming from biomasses. It was estimated (i.e. `msr1`) from the source `src1`, which is itself described through the URL and the creator. `ent1` is associated to the municipality `mun1`, which is located in the province `prv2`. This province is already represented in another dataset of the Web of Data. We explicitly indicate this equivalence through a link with the property `owl:sameAs`.

The example underlines our URI policy. We have specified URI structures for classes, properties and instances. The URIs for classes follow the pattern `http://www.openwatt.net/{concept name}` where `{concept name}` is usually the name of the class, e.g., `Estimation`. The URIs for properties follow the same structure `http://www.openwatt.net/{property name}`. The URIs for the instances are structured as `http://www.openwatt.net/{concept name}/{key}`. In this case, `{concept name}` refers to the class to which the in-

<sup>12</sup> Terse RDF Triple Language.

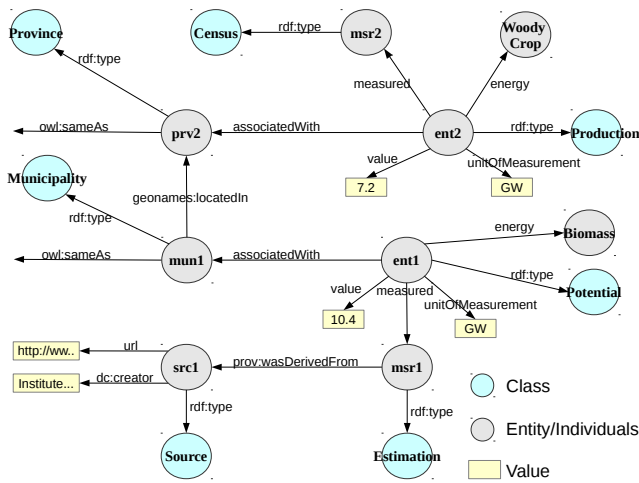


Figure 5: OpenWatt sample data graph.

stance belongs to and {key} denotes a unique alphanumeric identifier of the instance. If natural keys exist we use them<sup>13</sup>.

#### 4. IMPACT AND PERSPECTIVES

In this section, we try to highlight the potential impact and the civic relevance of OPENWATT. The whole Renewable Energy sector will benefit of more efficient, convenient and accurate tools.

The work to cleaning and integrating the data is repeatedly and redundantly made by everyone approaching the matter, but it never becomes a common asset, as everyone starts it from scratch every time. OPENWATT intends to convert data in standard representations, so that everyone can directly use the data together with a high quality semantic description of their. Another positive impact of OPENWATT is on the mobility of people working in the RE sector. On-the-spot inspections and displacements, the most frequent and time-consuming activities when planning, could be considerably limited.

Public organizations are actively involved in energy management of the territories. Many of them regularly produce local energy plans, where information on provision of primary energy sources, renewable and non renewable, and on their use and consumption are collected and provided. OPENWATT returns this important patrimony in an integrated way to the tax payers.

The benefits about quality and semantic interoperability of data will be effective with the involvement and participation of PAs, citizens and enterprises. PA opens itself to citizens for the sake of transparency and direct participation to decision making processes. Apart from counting on a massive quantity of data, individual citizens and associations could contribute in a crowd-sourcing fashion to collect more data. As a matter of fact, detection tools (e.g., anemometers, small weather stations, heliometers, etc.) are often provided free of charge in exchange of data to be collected by them. This enlarges the territorial density covered by OPENWATT.

<sup>13</sup> A natural key is a key that is formed of attributes that already exist in the real world, e.g., the post code for a municipality.

```

@prefix ow: <http://openwatt.net/> .
@prefix rdf:
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix prov: <http://www.w3.org/ns/prov#> .
@prefix
  geonames: <http://www.geonames.org/ontology#> .

<http://www.openwatt.net/Potential/ent1>
  rdf:type owl:Potential ;
  ow:energy owl:Biomass ;
  ow:measured
    <http://www.openwatt.net/Estimation/msr1> ;
  ow:associatedWith
    <http://www.openwatt.net/Municipality/mun1> ;
  ow:value "10.4" ;
  ow:unitOfMeasurement "GW" .

<http://www.openwatt.net/Estimation/msr1>
  rdf:type owl:Estimation ;
  prov:wasDerivedFrom
    <http://www.openwatt.net/Source/src1> .

<http://www.openwatt.net/Municipality/mun1>
  rdf:type owl:Municipality ;
  owl:sameAs <http://www....> ;
  geonames:locatedIn
    <http://www.openwatt.net/Province/prv2> .

<http://www.openwatt.net/Source/src1>
  rdf:type owl:Source ;
  dc:creator "Institute ..." ;
  ow:url "http://www..." .
...

```

Figure 6: OpenWatt sample RDF Data.

Indeed, the maximum achievement of the project would be making OPENWATT a universally recognized point of reference in the RE planning for the specialists and operators of the sector.

#### 5. IMPLEMENTATION

In this section we explain the technical work for implementing OPENWATT, providing implementation details of the phases referred in the previous section.

1. There are governmental and private energy databases used for planning purposes. At the moment of publication we are considering among the others: the Italian Atlas of solar radiation<sup>14</sup> that is useful for inferring potential energy from photovoltaic systems, the RSE eolic Atlas<sup>15</sup> containing shape files about wind measures, the National registry of zootechnics<sup>16</sup> containing details at municipal level about farming (e.g., pigs, cattle, horses, etc.), the GSE 2011 report on renew-

<sup>14</sup> <http://www.solaritaly.enea.it>

<sup>15</sup> <http://atlanteolico.rse-web.it>

<sup>16</sup> <http://www.izs.it/ZS/>

able energies<sup>17</sup> containing production plants of different categories divided into provinces.

2. We used existing techniques and tools (e.g., relational databases) for the cleaning. For instance, we checked the values of similar concepts, we checked if there were missing rows (e.g., missing provinces), etc.
3. The work of this step is explained in the Section 3.
4. We represented manually the parts of the datasets in order to check if the model of the previous step was suitable and could represent the data correctly.
5. The work in this step is explained in Section 3 and was conducted using the editor Protégé ontology<sup>18</sup>.
6. Since we have heterogeneous kinds of data, several different methods were adopted in this matter. To convert CSV, TSV, relational databases and spreadsheets, we used the framework D2RQ [6], which allows to flexibly define mapping from the tabular to the RDF model. In case of shape files and Web APIs, we extracted the data and stored them into relational databases. We used PostgreSQL 9.1 as relational database management system. In case of XML source data, we used XSLT programs.
7. Most of metadata were added manually. Some other were obtained through a transformation as it happened with the data.
8. The external links were produced by configuring scripts for SILK [19], a record linkage tool for Linked Data.
9. We defined a set of SPARQL queries to assess the logical validation. For example, we checked that production data were not related to estimation measures. To validate the syntax we used the available validators: the W3C validator for the data<sup>19</sup> and the validator from the University of Manchester<sup>20</sup> for the ontology.
10. We used the triple store Openlink Virtuoso<sup>21</sup> to load our data. It exposes, by default, a SPARQL endpoint.

To make more efficient this long process, we will think of products for the native production of linked on the Web data from sensors (i.e. Sense2Web [2]). For facilitating the development of applications by third parties, we are setting up an open platform to develop, deploy and run applications, with useful services to link and mash-up data. The platform will expose interfaces and provide standard APIs, allowing the integration with other tools or products by means of plug-ins. Moreover, we will release all the software as Open Source, GPLv3 licensed software, for boosting the process of community development.

## 6. CONCLUSION AND FUTURE WORK

In this paper we have presented OPENWATT, the Web of Data space containing renewable energy data. It overcomes issues related to accessibility, interoperability and understandability of data on the Web. In this way, everyone is allowed to build applications, both mobile and Web, that facilitate the integration with legacy informative systems.

<sup>17</sup> <http://www.gse.it/en/pressroom/News/Pages/statistical-report-2011.aspx>

<sup>18</sup> <http://protege.stanford.edu/>

<sup>19</sup> [www.w3.org/RDF/Validator/](http://www.w3.org/RDF/Validator/)

<sup>20</sup> <http://mowl-power.cs.man.ac.uk:8080/validator/>

<sup>21</sup> <http://virtuoso.openlinksw.com/>

As future work, we will take into account more data as we want OPENWATT to scale at international level. This is a very challenging task since there will be the need to collect and organize massive amounts of data. The evolutionary support needs to be well-designed. In addition, the ontology will be extended according to new concepts that we will come across and we will appropriately model. We are investigating the working modes to gather data from the crowd and we are planning to develop demo applications on our data.

## 7. REFERENCES

- [1] H. Abanda and J. H. M. Tah. Linked data in renewable energy domain. In *6th International Congress on Environmental Modelling and Software (iEMSs)*, 2012.
- [2] P. M. Barnaghi and M. Presser. Publishing linked sensor data. In *SSN*, 2010.
- [3] F. Bauer, D. Recheis, and M. Kaltenböck. data.reegle.info - a new key portal for open energy data. In *ISESS*, pages 189–194, 2011.
- [4] T. Berners-Lee. Linked data. <http://www.w3.org/DesignIssues/LinkedData.html>, 2013.
- [5] B. Bertin, V.-M. Scuturici, E. Risler, and J.-M. Pinon. A semantic approach to life cycle assessment applied on energy environmental impact data management. In *EDBT/ICDT Workshops*, pages 87–94, 2012.
- [6] C. Bizer. D2r map - a database to rdf mapping language. In *WWW (Posters)*, 2003.
- [7] P. Chaussecourte, B. Glimm, I. Horrocks, B. Motik, and L. Pierre. The energy management adviser at EDF. In *International Semantic Web Conference*, 2013.
- [8] F. Corno and F. Razzak. Publishing lo(d)d: Linked open (dynamic) data for smart sensing and measuring environments. In *ANT/MobiWIS*, pages 381–388, 2012.
- [9] M. Cotterell, J. Zheng, Q. Sun, Z. Wu, C. Champlin, and A. Beach. Facilitating knowledge sharing and analysis in energy informatics with the ontology for energy investigations (OEI). *Energy Informatics*, 12(4), 2012.
- [10] G. Lodi, A. Maccioni, and F. Tortorelli. Linked open data in the italian e-government interoperability framework. In *6th International Conference on Methodologies, Technologies and Tools enabling e-Government (METTEG)*, 2012.
- [11] A. Maccioni. Towards an integrated social semantic web. In *2nd International Workshop on Data Management in the Social Semantic Web (DMSSW)*, 2013.
- [12] A. Miles and S. Bechhofer. Skos simple knowledge organization system reference. Technical report, W3C, 2009.
- [13] P. Missier, K. Belhajjame, and J. Cheney. The w3c prov family of specifications for modelling provenance metadata. In *EDBT*, pages 773–776, 2013.
- [14] T. B. Pedersen, W. Lehner, and G. Hackenbroich. Report on the first international workshop on energy data management (endm 2012). *SIGMOD Record*, 42(1):50–52, May 2013.
- [15] F. Razzak, D. Bonino, and F. Corno. Mobile interaction with smart environments through linked data. In *SMC*, pages 2922–2929, 2010.
- [16] D. Rech and A. Harth. Towards a decentralised hierarchical architecture for smart grids. In *EDBT/ICDT Workshops*, pages 111–115, 2012.
- [17] L. Sauer mann, R. Cyganiak, and M. Völkel. Cool uris for the semantic web. 2011.
- [18] S. Teske, T. Pregger, S. Simon, T. Naegler, W. Graus, and C. Lins. Energy [R]evolution 2010: a sustainable world energy outlook. *Energy Efficiency*, 4(3):409–433, 2011.
- [19] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk - a link discovery framework for the web of data. In *LDOW*, 2009.



# A Generic Ontology for Prosumer-Oriented Smart Grid

Syed Gillani  
Telecom Saint Etienne,  
Université Jean Monnet  
Saint Etienne, France  
syed.gillani@univ-st-  
etienne.fr

Frederique Laforest  
Telecom Saint Etienne,  
Université Jean Monnet  
Saint Etienne, France  
frederique.laforest@telecom-  
st-etienne.fr

Gauthier Picard  
Institute Henri Fayol, EMSE  
Saint Etienne, France  
gauthier.picard@emse.fr

## ABSTRACT

The concept of Smart Grid (SG) has comprehensively overhauled the scene of electric power grid and the integration of *Prosumers*, where an entity can consume and produce simultaneously extemporised in a complete paradigm shift. This requires a detailed knowledge base model at each entity level that can react according to contextual changes.

This paper outlines a generic and layered ontology design for such complex *Prosumer* oriented SG, which enables the autonomous integration and real-time management of distributed and heterogeneous sources. It details the relevant layer to the right granularity with keen insight into distributed co-ordination and semantic heterogeneity. It also presents an inductive based reasoning on such ontology to make it thoroughly elucidative.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous

## Keywords

Smart Grid, Prosumers, Ontology design

## 1. INTRODUCTION

Electrical Grid has evolved surprisingly less over the past 50 years, while the population and electricity demand has grown considerably. The current version of electrical grid suffers from number of problems including, inefficiency, unreliability in demand response, ill-equipped to handle integration of renewable sources and relying on uninformed infrastructure to educate users regarding their usage level at any given time. The term Smart Grid (SG) can be illustrated as according to the description of Energy Independence and Security Act of 2007<sup>1</sup> "Smart Grid refers to the modernization of electricity delivery system that can monitor, protects and automatically optimizes the operations of its interconnected elements.

<sup>1</sup><http://energy.gov/eere/femp/articles/energy-independence-and-security-act>

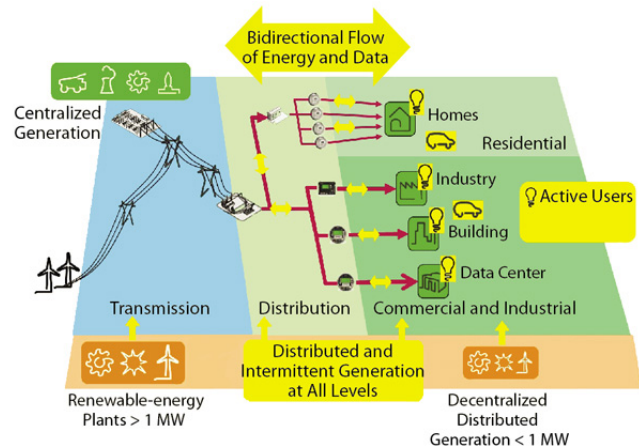


Figure 1: Prosumer Oriented Smart Grid<sup>2</sup>

It starts from central and distribution generation, through the organization of system, to the end user's (residential, industrial) automation, electricity storage and household devices". Hence, SG is characterized by distributed flow of energy and information to create an intelligent energy delivery system.

This paradigm shift introduces an intriguing concept of *Prosumers*, an entity that blurs the distinction between the consumer and producer as depicted in Figure 1. Such docile combination allows a bi-directional flow of energy and information, where even the consumers (residential, industrial) can produce and trade unused energy. The centralized grid in-charge of distribution can utilize this reverse flow of energy by herding it towards power critical users. Moreover, geographical localized Prosumers can conform a Smart micro grid to share over-harvested energy. This accrues systematic reliability of SG and decreases the cost, while eschewing CO<sub>2</sub> emissions by employing renewable energy sources at Prosumer premises. This cohesion of efficiency by such system can be regarded as passive manoeuvre by engaging a consistent consumption model, while Prosumer oriented SG triggers an active plan by manifestation of Demand Response (DR), thus allowing the production of energy on the basis of meticulous demand, which in return mitigate the energy losses.

Engineering an intelligent Prosumer oriented SG requires obsequious modelling of information acquired from distributed sensors, while considering the realm of both producers and consumers. Although, the main challenge in this case is, how to collect data from

<sup>2</sup><http://www.greenmanufacturer.net/article/machinery-and-equipment/demand-response-the-power-program-that-pays-you-back>

each energy source at individual level, considering the presence of various techniques and data models. The two indigenous approaches used to acquire such data are *Direct Sensing* and *Single Point Sensing* [5]. *Direct Sensing* employs each device with a distant sensor, while *Single Point Sensing*, uses one sensor to disambiguate all the appliances with total voltage measurement as events [6]. Since Prosumer oriented concept of SG involves in profound role change of system-wide stakeholders (Consumers, Producers, Prosumers), there is a compelling need of handling, storing and extracting knowledge out of data received from heterogeneous sources. Semantic web offers such functionality by allowing Smart Grid to imbue with intelligence, logical and domain specific reasoning and meticulous mapping between entities. Thus, a generic and layered ontology plays a vital role in embedding efficient knowledge reasoning and management of bi-directional flow of information. The rest of the paper is distributed into two sections. Section 2 discusses the design and requirements for generic and layered ontology and Section 3 examines the rule-based inductive reasoning on domain ontology.

## 2. ONTOLOGY MODELLING

An optimized ontology is characterized by encapsulating entire domain concepts to the right granularity. Since modern systems evolve continuously with time and SG is still far from its standardized model, thus it faces number of issues, such as complexity of network, large number of entities, diversity in architecture and highly flexible requirements. This solicits a generic knowledge base design, where all the participants of the system are modelled and linked to the right granularity. The idea of dynamic ontology integration [8] may be applicable to such context, where different versioning techniques are used to focus the impact of changes on logical consistency of ontology. Although, in the context of complex Smart Grid, where current and historical information are both expedient and system comprised of large number of heterogeneous and distributed sources, such techniques won't suffice to solve semantic heterogeneity [9]. For instance a media player could be a simple stereo system or a professional sound system. This however, enlightens the pressing need of a generic and layered ontology design, where these diverse entities should be mapped according to their context. The acceptable requirements of such system are two folded. Firstly, the knowledge base data model should encapsulate all the information at right granularity (generality). Secondly, it should entertain any evolved context without disturbing the overall structure of knowledge base (layered). Therefore, it requires an ontology design that can adapt according to a new concept, such as if a new device is added at consumer premises that is not yet defined in domain ontology. Furthermore, the system should employ a layered ontology, where classified concepts could be modified without affecting the whole model. For instance, if a new type of power generator is added to the system, it should only require the addition of new concept in right class without modifying any link in domain ontology. The integration of Prosumers to such complex system, where individual entities co-operate with each other according to their goals and geographical location, begets it more challenging to annotate available data with well-defined relationships that provide the accurate context of their use.

### 2.1 Use Cases

In order to consolidate and excite the idea of modelling multi-domain and flexible ontology, we present some use cases as following. This is not an exhaustive list, but a cross-section of interesting use cases.

#### Ad-hoc Based :

- A-UC 1 Find a residential sector with highest power consumption rates and advise the system regarding the type and number of new storage or renewable sources that should be added in order to satisfy its demands and reduce energy cost (micro grid composition).
- A-UC 2 Determine the most economical, reliable and environmental friendly energy producer in a certain city. What kind of energy is produced by such source and compare the change in their production capacity with corresponding weather conditions?
- A-UC 3 Determine the number of consumers connected to a specific producer and list their infrastructure type, energy classes and total revenue generated since last 2 months.
- A-UC 4 Compare all the power sources used by a certain premises for the last 6 months as according to their reliability, costs and environmental effects.

#### Advice Based :

- AB-UC 1 What is the total power consumption for appliances with power rating greater than 1000 Watt in a certain premises and describe their usage patterns for the last month at the temperature range of 27-35 C. Additionally, advise about better scheduling to reduce the cost of electricity?
- AB-UC 2 List feasible dates and times, which would be cost effective to operate a washing machine in certain premises by comparing past consumption patterns.

#### Event Based :

- E-UC 1 In an event of power failure or shortage from a certain generation source, switch to back-up energy storage or other available power sources.
- E-UC 2 If the generator power source rating is 3KWatt for last 2 hours during peak time with price ``P1" and only low voltage (10-100Watt) appliances are operational at current time, then check the status of attached power storage unit, if it's more than 70% charged then shift the load from the main power source to storage source.

The following section illustrates our ontology design for Prosumer oriented SG. Our ontology<sup>3</sup> is divided into eight conspicuous layers as depicted in Figure 2.

### 2.2 Infrastructure Type

The power consumption patterns directly correlate with infrastructure's operational type, time and geographical location, which evokes the requirement of such classifications. For instance a residential house will consume more energy at night or weekends, while consumption demand of an office varies according to working hours. Additionally, these premises based classification can identify each entity in the system by using the address property and attributes it as normal or power critical, e.g. a hospital requires more reliable energy source as compare to residential property. These distributions are influenced by UK property classifications<sup>4</sup> and are as following.

**Commercial Premises** Entities in this class consist of commercial premises, such as retailing shops, food restaurants, cinemas etc. with varying operating times depending on the type.

<sup>3</sup><http://data-satin.telecom-st-etienne.fr/ontologies/smartgrids/smartgrid2.ttl>

<sup>4</sup><http://www.legislation.gov.uk/ukxi/1987/764/article/3/made>

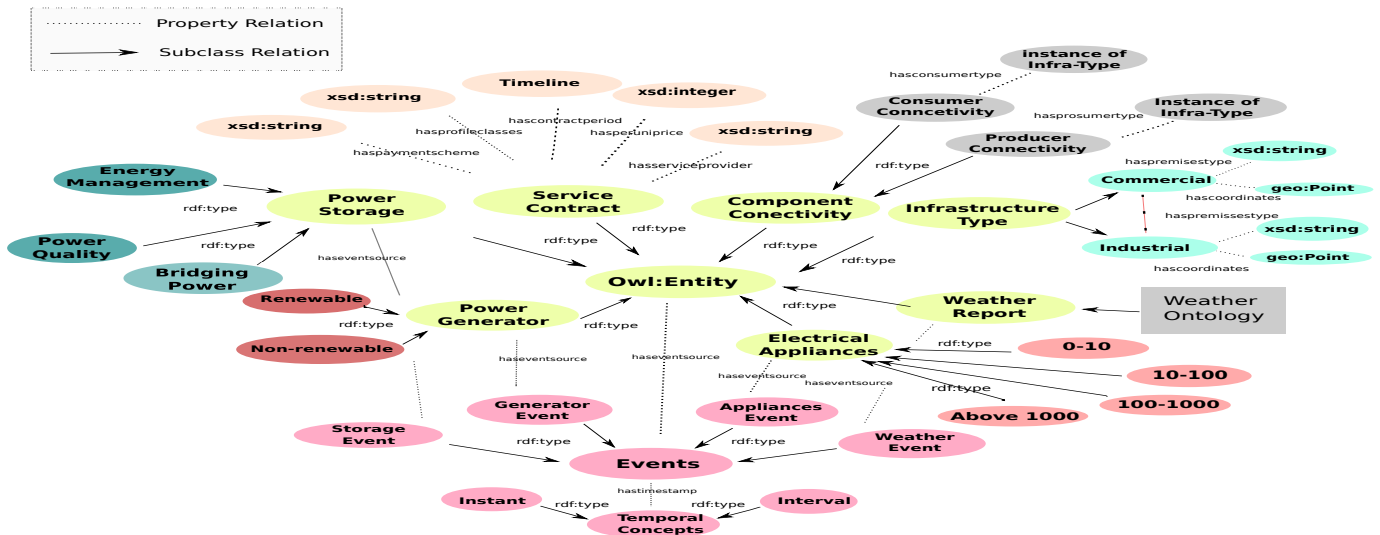


Figure 2: Prosumer-oriented SG Ontology Design

**Business Related Premises** This class constitutes various business related entities. These premises usually operate during office times, thus it would permit to postulate their consumption behaviours and better scheduling algorithms.

**Residential Premises** This class deals with residential premises and plays a vital role to determine the overall consumption of the SG, as it constitutes of nearly quarter of total energy consumption, additionally, residential consumption patterns are agitated before and after office hours.

**Non-residential Institutions** This class consists of non-residential institutes such as hospitals, schools, libraries etc. These premises accord as power critical ones and require a higher priority.

**Industrial Premises** This class defines the industrial infrastructure, where industries demand an uninterrupted and reliable power source, thus modelling such entity portrays the consumption pattern of power critical sources and aids in assigning power sources.

The importance of such distribution can be concluded from [A-UC 1]. Hence, in order to accurately advise a consumer reading storage choice, infrastructure type should be taken into consideration because residential premises can't afford an industrial type storage unit and its preferred choice should have fast charging and discharging response. Furthermore, micro grid composition is more feasible in residential sector as compared to industrial one. Such information comfort the composition of smart micro grid, better scheduling to reduce consumption costs and prioritise the power distribution accordingly.

### 2.3 Electrical Appliances

The efficient management of heterogeneous electrical appliances residing in different entities require automatic collection of data from power consumers. Direct sensing promises an exact consumption and temporal data, while single point sensing reasoned for a more practical solution by using a single sensor to disambiguate different appliances. This demands a generic data structure that can accommodate both direct and single point sensing. Additionally, there could be numerous novel devices tuning in to the system that would require a flexible data structure. This leads us to classify these devices in a two layered architecture, where first layer divides the consumer devices according to their power ratings, while second

layer classifies them according to their exact type (mining appropriate IEC CIM<sup>5</sup> defined devices). This enables the ontology to grasp the dynamic nature of SG.

The gravity of this distribution can be judged from [AB-UC 1]. Where appliances with higher or mid-level power consumption requires better scheduling and most of these appliances (washing machines, water heaters) demonstrate repeatable switching pattern along certain time periods, while appliances with lower levels permit to continuous consumption pattern. Consequently, these classifications will aid in implementing inductive inference to distribute appliances according to their operational/consumption patterns. This assortment of appliances are comprised of 0-10 Watts of basic household and office appliance with lower priority of scheduling, 10-100 Watts of mid range appliances with lower priority of scheduling, and over 1000 Watts of higher powered and priority devices.

### 2.4 Electrical Generation System

Smart Grid encompasses a vast number of diversified, distributed and variable energy sources, which endeavour to model these energy sources within the domain ontology. Additionally, the comeliness of Smart Grid resides with its flexible architecture that allows the integration of renewable energy sources, while mitigating the climate effects. This very point rested a compelling case to classify each power source not only according to its operational type but also regarding the type of produced energy. This classification will provide an omniscient view of the system, while offering valuable information to the end users. Our ontology captures these classifications as following.

**Non-Renewable Energy** Non-renewable energy sources encompass all kinds of fossil fuel and its classification is based on their carbon footprint, while considering both direct - arising during operational power plant and indirect - arising during the non-operational phases of production life cycle (cradle-to-grave).

**Renewable Energy** Renewable energy exists perpetually, inexhaustible and more importantly a clean alternative to non-renewable sources. Although it is regarded as green energy, but its reliance on weather conditions demands the classification of renewable sources according to their operational types. For example wind, solar.

<sup>5</sup><http://www.iec.ch/smartgrid/standards/>

The classification of such power sources can be regarded as an important part of the ontology and its absence would impede efficient management of energy sources. [A-UC 2] enlightens its importance. For instance, a competent energy source should be reliable and environmental friendly, where fossil fuel produced energy is regarded as reliable with huge  $CO_2$  footprint, while renewable energy sources are much more cleaner but its reliability varies according to geographical locations and weather conditions.

## 2.5 Power Storage System

Power Storage systems provide three crucial services, firstly it depletes the electricity cost by storing it during the off-peak times, secondly, it improves the reliability of the system in case of power network failure. And finally to maintain and improve the power quality, frequency and voltage. Additionally, renewable energy sources can maximise their production efficiency when paired with an energy storage system. These storage systems are classified according to type, produced power, charge-and-discharge efficiency, cost per KWH [3]. Such attributes can guide the consumer regarding economics and reliability of a storage system. There are two principle criteria to classify various storage systems: functional and forms, as the context of SG demands more of functional attributes, consequently modelling such behaviours at knowledge level resonates well with load balancing and scheduling of power storage systems.

**Energy Management** These types of storage solutions are usually equipped with large-scale entities drawing power up to 100MW. Hence, they can be regarded as sole power generator sources and most of the industrial based renewable energy sources are connected to such storage units. This class of devices is attributed with higher charging and discharging time periods.

**Power Quality** Storage devices of this class are utilized for power quality, such as instantaneous voltage drop, flicker mitigation and short duration uninterrupted power supply. They are equipped at relatively lower power consumers (residential) and have lower charging and discharging time periods.

**Bridging Power** These are more responsive type of storage devices with relatively fast response and long discharging time. These are usually installed at residential renewable sources with power rating of about (100KW-10MW).

This distribution recommends an efficient scheduling of power generator and storage sources and [A-UC 1] is influenced by the right characterisation of these storage sources. As the right functional type of these sources determines if a premise is using it merely as an energy source or as a dis-patchable generation source to accord for power shortage and to mitigate frequency problems. Furthermore, by determining weather conditions, infrastructure type and required characteristic of an entity, system can predict a better choice of storage unit. This constitute a strong case for behavioural distribution of storage units rather than just according to their storage capacity.

## 2.6 Weather Report

Weather and temperature are important drivers of electricity consumption and production patterns and meticulous forecast of energy production relies on accurate measurement and modelling of weather data. [AB-UC 1] and [A-UC 2] straighten out the importance of such information. As change in weather conditions leads to the use of high power appliances (air-con in summer, heater in winter) and in order to avoid power shortage during these periods, the system should build forecasting models for consumption patterns and schedule these sources in acceptable and efficient manner. Furthermore, the production of alternate or renewable energy directly

relates to fickle weather conditions and reliability of these sources can be predicted by modelling weather related information. There are various ontologies available for modelling such functionalities, such as NNEW<sup>6</sup> and SSN<sup>7</sup>.

## 2.7 Events

Smart Grid deals with real-time monitoring and management of interconnected entities, this emanates a system equipped with Complex Event Processing (CEP) [1] that is able to detect occurrence of specified patterns of events and respond accordingly. Thus, fabricating events becomes a legitimate objective of ontology, where it must be capable of accounting for spatial relation both synchronically and diachronically. Such functionality can be embedded to ontology by classifying each event type, temporal annotation and their relationship with domain entities. This distinction is based on the fact that an event may involve with number of processes. E.g. a change in the consumption power of an entity might involve in turning a washing machine off or turning on a hover. This requires an exact relationship of an event with corresponding sub-entity. Our ontology segregates events into following four types.

**Electrical Appliance Events** These events are coined to the electrical appliance and trigger the change in the consumption pattern, which in result prompt the process of load scheduling or change in demand response.

**Weather Event** These events capture the context of drastic change in weather, as power consumption and production is directly related to weather condition, thus mining those events lay down the ground for predicting consumption and production behaviours.

**Storage Events** The anticipated production capability of a storage system is highly dependent on its temporal aspects, such as, performance of a storage unit is directly proportional to its charging and discharging time and complying with such events aid in articulating the reliability of certain premises.

**Generator Events** These events deal with the power production capabilities of producers. For instance, if there is a change in the production capabilities or failure of a generator.

Consequently, the notion of Prosumers consolidated with renewable energy sources portray a more complex glimpse of the system as depicted in [E-UC 1] and [E-UC 2], where with frequent negotiations between available producers crave to capture the current relationship between consumers and producers. Furthermore, in order to predict the reliability of such production facility the temporal events linked to the corresponding sources must be added to the domain ontology.

## 2.8 Service Contracts Ontology

Service contracts or agreements are legally enforceable promise or undertaking along-with associated conditions. In case of Prosumers, these contracts will be between two parties, one who produced electricity (seller) and one who will like to consume electricity (buyer). Contractual information is quite cogent for communication between producer and consumer domains as in a competitive market; this information will directly cajole the decision and preference of both parties. Modelling such information will enable the consumer to choose a service provider entailing better economical deals and will guide producer to offer lucrative deals in order to attract maximum number of customers. The various properties associated with this class of ontology includes, the name of service provider, Start Date/End Date of contract, Profile Classes<sup>8</sup> of con-

<sup>6</sup><https://wiki.ucar.edu/display/NNEW/>

<sup>7</sup><http://www.w3.org/2005/Incubator/ssn/ssnx/weather-station/>

<sup>8</sup><http://www2.ademe.fr>

nected consumer, type of payment scheme for the connection, early contract termination charges and per unit price of electricity. This part of ontology draws a comprehensive attention towards competitive energy market. Consequently, with the integration of Prosumers, each entity will prefer a flexible energy package and such properties in domain ontology not only assist consumer to determine the right connectivity type but also advocate a producer to predict the approximate energy demand of a consumer according to described consumer profiles and thus, calculate total revenue generated from such consumers as depicted in [A-UC 3]. For example, an energy provider can classify a city according to consumption classes of customers while calculating revenue in each sector and prioritise or increase the production facilities to areas with higher revenues or predicted energy consumption.

### 2.9 Component Connectivity

The movement towards renewable energy resources and the apparent awareness of energy consumption lead to new challenges in the distribution grid and energy production. This distributed and decentralized power generation leads to the concept of smart microgrid, where a group of loads and energy sources are aggregated together to appear as a single asset in a localized way. This formation requires the correct location of connected sources, thus this part of the ontology focused on defining the exact connectivity relationships between producers and consumers. Furthermore, these properties can also be used to determine the efficiency and expected load of consumer entities. An entity coined as a power source classifies consumers according to their total consumed power and an entity under consumption's tutelage record the type of energy produced by the power source and total power consumed from such sources, while inheriting all the types discussed in Power generation and Storage system. Component connectivity advocates much of the moral motivations of a consumer. Since consumer decides its energy source according to its reliability and produced carbon footprint. As depicted in [A-UC 4], the connectivity information of these distributed sources and sinks is quite essential to compare the overall performance, such information assist in reasoning the user's preferred energy type, as environmental aware consumers will prefer alternate energy, while industrial users will prefer more reliable energy source like nuclear or fossil fuel.

## 3. RULES BASED INDUCTIVE INFERENCE

Inductive inference involves in looking various patterns/trends and classifying them according to their properties. Its judgement process is influenced by heuristics and rules that tap into associative information about context and similarity. This, however is more non-monotonic in nature, where the conclusion of premises are drawn much due to the presence or absence of them and are bound to change, when more knowledge is acquired and previously drawn conclusion may have nullified, as the rules of inference that led to them may no longer be active. Such inference process consist of two main approaches, where the first one merely draws statistical conclusion based on historical data, while the later deals with predicting future values by utilising probabilistic models, such as Bayesian inference [4]. Let  $X$  is a set of observed characteristics from data and  $Y$  is a set of predicted outcomes then,  $\Omega = (X \otimes Y)^\infty$ , where  $\Omega$  is a set of concluded states and can be narrow down to the right granularity by employing Bayesian model [4]. In the context of SG, where efficient scheduling and management of energy sources makes quite the case, these inferred incremental part of ontology drives the effective reasoning and analytical job. The three main types of these incremental ontologies

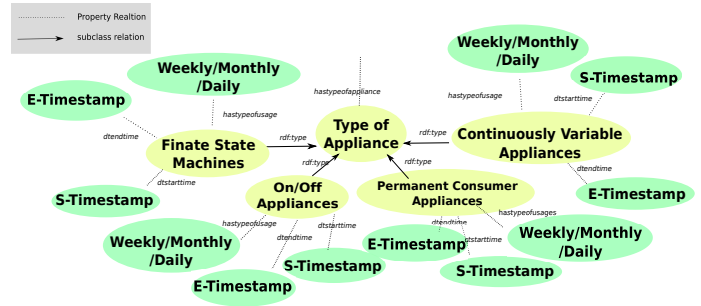


Figure 3: Inductive Ontology for Appliance Consumption Patterns

that ex-cogitate from inductive inference are as following.

### 3.1 Appliance Consumption Patterns

The efficient management of electricity requires classification of consumer appliances according to their operational patterns and provide the required flexibility in controlling electricity costs. Such methods of controlling electricity consumption are directly related to demand response, which relies on varying price of electricity to reduce peak demand. For instance, a washing machine electricity consumption pattern is usually weekly, hence it can be scheduled to operate at night during weekdays ([AB-UC 2]). Furthermore, consumer entity could entail the production attribute, thus it can sell its produced power according to the consumption patterns of its appliance. Consumer appliance can be classified according to assigned rules into three main types, depending on their consumption patterns [10] [6], as depicted in Figure 4.

**Finite State Machines or Multi State Devices** This category includes the devices that have repeatable switching patterns. Thus, the switching cycle could be repeated daily, weekly or monthly. Examples are washing machines, dryers and lawn mowers.

**On/Off Devices** This includes the most household devices that are turned on and off in frequent manner, such as toaster, light bulb, water pumps etc.

**Permanent Consumer Appliances** This category includes the appliances that remain operative for all the time (24x7). Example includes hard-wired smoking alarms and some power supplies.

**Continuously Variable Consumer Devices** This category includes the devices with variable states and draw power randomly without any specific pattern. E.g. Power tools, dimmer lights.

These classifications follow the non-monotonic principle and they revised itself with change in newly acquired data. Furthermore, as according to [AB-UC 1] and [AB-UC 2], extracting meaningful appliance patterns can not only assist the users with understanding their behaviours but also to make judgement in relation to better scheduling of appliances during peak hours in order to reduce the overall cost of electricity.

### 3.2 Alternate Energy Production Patterns

Energy produced by almost all renewable energy technologies is wholly dependent on the weather. Wind and wave power depend on the speed, direction and duration of the wind. Solar power, whether photovoltaic or thermal, depends on the intensity and duration of solar irradiation. Consequently, weather and climate is a common



denominator for all of these increasingly important sources of renewable energy. To determine the expected energy yield of a renewable power source, inductive reasoning can match the past patterns (production and weather) and predict the future production accordingly. Embedding this information in the ontology will enable the consumer to predict the reliability of the energy source and also helps in distributing load of consumer appliances. IBM's<sup>9</sup> new advanced power and weather modelling technology showcases the importance of such information.

### 3.3 Producer's Performance Patterns

The performance of an energy producer can be judged from its efficiency, economical operations and impact on the environment. SG not only promises a more efficient and reliable system but also emphasis on the reduction of greenhouse gases (GHG), which emphasis on reliable audit of energy producer. Hence, each consumer can infer the reliability of its producer by applying inductive reasoning on its past patterns. As performance critical entities, for instance hospitals and industries will prefer a more reliable source. Furthermore, in order to accurately compare carbon footprints of these different technologies, the total  $CO_2$  emitted through the production cycle must be calculated. This measurement will influence the consumers to priorities the choice on the bases of reliability and/or environment friendly consumers. The carbon footprint of each producer entity can be calculated by comparing the type of the power plant and produced power.

## 4. RELATED WORK

There are various industrial standards that co-exist for SG model, e.g. IEC CIM and NEMA<sup>10</sup>, but these standards are not diversified, and detailed enough to embed into Prosumer oriented SG. Although appliance description as mentioned in IEC CIM (as discussed in Section 2.3) is quite interesting and our model extracts and extends these useful concepts and exploits them for low-level appliance description.

Semantic based modelling discussed in [7] offers certain design guidelines to effectively integrate smart grid to the household appliance. It emphasis on heterogeneous data acquisition and how it could be used to provide value added services to users. But, with the integration of distributed alternate energy resources and Prosumers, the model requires to cater this useful information to harness the effective energy exchange between various entities. [11] describes an interesting ontology for mapping household devices by integrated various available ontologies. It also outlines an interesting event processing architecture considering the context of Smart Grid. The main drawbacks that can be inferred from such techniques are its reliance on direct sensing and failure to capture real world scenarios. Its ontology is quite detailed in nature by integrating IEC CIM concepts, where such information cannot be extracted from each and every household. Furthermore, nothing sustainable has been discussed regarding the compatibility of these integrated ontologies and presented ontology is still in conceptual stage without any implemented version. [2] describes a semantically rich energy management system, where ontologies are used to represent each customer in a relevant domain, focusing on its energy usage and environment. These facts are then reasoned to infer the relevant tips for customers. This work doesn't detail any information regarding the acquisition of data and the conceptual ontology and reasoning is performed on a static set of triple store rather processing any real time events. However, A-box and T-box assertion from

customer data mimics the usefulness of extending ontology in the context of energy management systems. In a whole, state of art ontologies for Smart Grid are unable to capture the real world scenarios or various heterogeneous domains (generation, storage) this leads to an abstract ontologies that doesn't cater each domain level in detail.

## 5. CONCLUSION AND FUTURE WORK

In this paper we present a multi-dimensional and generic ontology model equipped with inductive based reasoning and complex event processing, by attributing each domain of interest with relevant relations. These relations and requirements, as depicted in our ontology are ratified through our use cases, while considering the context of Prosumer oriented SG. Our future endeavours involve in integration of proposed ontology with SEAS project<sup>11</sup> and regress testing with various SG based simulators. Furthermore, we intend to develop an open source engine for complex event processing using distributed multi-agents paradigm in the context of Prosumer oriented SG.

## 6. REFERENCES

- [1] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic. Etalis: Rule-based reasoning in event processing. *Semantic Web: Interoperability, Usability, Applicability*, 2011.
- [2] P. Chaussecourte, B. Glimm, I. Horrocks, B. Motik, and L. Pierre. The energy management adviser at edf. In H. Alani, L. Kagal, and A. Fokoue, editors, *The Semantic Web-ISWC 2013*, volume 8219 of *LNCS*, pages 49--64.
- [3] H. Chen, T. N. Cong, W. Yang, C. Tan, Y. Li, and Y. Ding. Progress in electrical energy storage system: A critical review. *Progress in Natural Science*, Mar. 2009.
- [4] M. Feldkircher. Forecast combination and bayesian model averaging - a prior sensitivity analysis. Working Papers in Economics and Finance 2010-14, Sept.
- [5] J. Froehlich, E. Larson, S. Gupta, G. Cohn, M. Reynolds, and S. Patel. Disaggregated end-use energy sensing for the smart grid. *Pervasive Computing, IEEE*, 10(1):28--39, 2011.
- [6] G. Hart. Nonintrusive appliance load monitoring. pages 1870--1891. *IEEE*, 1992.
- [7] A. Monacchi, D. Egarter, and W. Elmenreich. Integrating households into the smart grid. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 2013.
- [8] P. Pittet, C. Nicolle, and C. Cruz. Guidelines for a dynamic ontology - integrating tools of evolution and versioning in ontology. *CoRR*, abs/1208.1750, 2012.
- [9] M. Uschold and M. Grüninger. Ontologies and semantics for seamless connectivity. *SIGMOD Record*, 33(4):58--64, 2004.
- [10] M. Zeifman and K. Roth. Nonintrusive appliance load monitoring: Review and outlook. *Consumer Electronics, IEEE Transactions on*, 57(1):76--84, 2011.
- [11] Q. Zhou, S. Natarajan, Y. Simmhan, and V. Prasanna. Semantic information modeling for emerging applications in smart grid. In *Proceedings of the 2012 Ninth International Conference on Information Technology - New Generations*, ITNG '12, pages 775--782. *IEEE Computer Society*.

<sup>9</sup><http://www.ibm.com/press/us/en/pressrelease/41310.wss>

<sup>10</sup><http://www.nema.org/pages/default.aspx>

<sup>11</sup><https://itea3.org/project/seas.html>

# Computing Electricity Consumption Profiles from Household Smart Meter Data

Omid Ardakanian  
University of Waterloo  
Waterloo, Ontario, Canada  
oardakan@uwaterloo.ca

Negar Koochakzadeh<sup>\*</sup>  
Oracle  
Vancouver, BC, Canada  
negar@koochakzadeh.net

Rayman Preet Singh  
University of Waterloo  
Waterloo, Ontario, Canada  
rmmathar@uwaterloo.ca

Lukasz Golab  
University of Waterloo  
Waterloo, Ontario, Canada  
lgolab@uwaterloo.ca

S. Keshav  
University of Waterloo  
Waterloo, Ontario, Canada  
keshav@uwaterloo.ca

## ABSTRACT

In this paper, we investigate a critical problem in smart meter data mining: computing electricity consumption profiles. We present a simple, interpretable and practical profiling framework for residential consumers, which accounts for variations in electricity consumption at different times of day and at different external temperatures. Our approach is to isolate the effect of external temperature on electricity consumption and apply a time-series autoregressive model to the remaining signal. The proposed profiles may be used for making personalized energy-saving recommendations, detecting outliers, and generating very large realistic data sets for testing the scalability of smart meter data management systems. Using predictive power as a metric for the accuracy of consumption profiles, we show, using a real data set of 1000 homes, that our approach results in improved root-mean-squared prediction error compared to existing approaches.

## 1. INTRODUCTION

Smart electricity meters are rapidly replacing conventional meters in many parts of the world. Smart metering systems offer many operational advantages for energy utilities and policy makers, including

- enabling automated collection of fine-grained (typically half-hourly or hourly) consumption readings, thereby eliminating the need for utilities to send out estimated bills or to dispatch personnel to customer premises and manually read the meters,
- enabling dynamic pricing schemes that depend on the time-of-day in order to reduce demand for electricity

<sup>\*</sup>Work done while the author was a Postdoctoral Fellow at the University of Waterloo.

during peak times.

However, exploiting smart metering systems to their fullest also requires mining the vast amounts of collected consumption data to obtain insights into grid operations and consumer behaviour [2, 4, 7, 12, 16, 21].

In this paper, we address the problem of computing electricity consumption profiles from smart meter data, with a focus on residential customers. The residential sector contributes a significant fraction to the total electricity demand (30 percent in Canada [5]) and greenhouse gas emissions (see, e.g., [19, 24] for United States statistics). Furthermore, in many regions, residential consumers are significant contributors to peak demand; e.g., in Ontario, Canada, residential air conditioning load is a major contributor to peak demand, which occurs in the afternoon of hot summer weekdays [23].

We argue that consumption profile generation is a fundamental smart meter data mining operation that electricity providers, resellers and consultants can perform, with at least the following applications:

- Conducting “virtual energy audits” and making personalized recommendations for saving electricity based on the trends identified in the profiles.
- Clustering households based on the features captured by the profiles. This may be used to understand different classes of consumers and to design targeted energy conservation and peak reduction programs for different classes.
- Generating real-time alerts if new consumption readings do not match the expected consumption predicted by the profiles. A related application is to identify consumers with “suspicious” load profiles that do not fit in any cluster, which could indicate electricity theft, malfunctioning meters or the presence of specialized equipment such as electric vehicle chargers.
- Generating realistic synthetic data based on the available real data. This may be used as input to grid simulation, transformer sizing, forecasting and pricing models, or to create very large realistic data sets for testing the scalability of smart meter data management systems.

## 1.1 Challenges and Contributions

In order to be useful for the above applications, we argue that consumption profiles must satisfy the following criteria.

- First, they must be *accurate*, i.e., able to describe and predict consumption with reasonable accuracy.
- Second, they must be easily *interpretable*; a complex machine learning model may be accurate, but if it is not interpretable, then actionable energy-saving recommendations cannot be easily inferred from it.
- Third, they must be *practical*, and therefore they should only require data that are easily available to utilities, such as hourly smart meter readings and weather. While household characteristics (e.g., home size and age, number of appliances, number of occupants, etc.) and consumer demographics could be useful, this information is typically not available to utilities due to privacy regulations and cannot be easily obtained without intrusive measurements and questioning.

The challenge in computing accurate and interpretable profiles from smart meter data is that residential electricity consumption depends on many factors, including the time of day, weather and the occupants' daily routines. Broadly speaking, prior work can be divided into two approaches. One is to compute various aggregate statistics from historical consumption data that account for typical daily activity; examples include the average, maximum, minimum and variance of hourly or daily consumption, ratios of night-to-day or morning-to-evening consumption, or identifying the hour of day when peak consumption usually occurs. The other approach has been to correlate consumption with external temperature, e.g., using piecewise linear regression, and use the correlation coefficients as representatives for the cooling and heating efficiency of homes.

In this paper, we propose a simple and practical technique that combines the best features of existing methods, and accounts for both temperature and activity in an accurate and interpretable fashion. The idea is to remove the effects of external temperature and outliers from the raw consumption data<sup>1</sup>, and compute typical hourly consumption values from the remaining signal using a time series auto-correlation model. This gives us typical consumption levels of a given home at different times of the day, independent of temperature and robust to "noise" (e.g., time periods when the home was empty or unusually busy).

Specifically, we make the following contributions in this paper:

- We propose a simple and interpretable technique for computing electricity consumption profiles from household smart meter data, which may be used for personalized recommendations, forecasting, classification, and as input to simulation models.
- Using predictive power as a metric for accuracy and hence the representativeness of consumption profiles,

<sup>1</sup>Here, by outliers we mean consumption readings that are much lower or higher than the average consumption for the given home, and thus do not correspond to the typical level of activity in this home.

we compare the proposed method to several existing approaches. Using a real data set, consisting of a year of hourly smart meter readings from 1000 homes in southern Ontario, Canada, we show that our approach outperforms existing approaches in terms of the root-mean-squared prediction error.

## 1.2 Roadmap

The remainder of this paper is structured as follows. Section 2 gives the intuition and an overview of our solution; Section 3 presents the details of our consumption profile algorithm; Section 4 describes our experimental results; Section 5 discusses related work; and Section 6 concludes the paper and discusses open problems in smart meter data management.

## 2. INTUITION AND SOLUTION OVERVIEW

In this section, we give an overview of our solution and we explain the intuition behind it. The input to our problem consists of two time series: 1) periodic (e.g., hourly) timestamped electricity consumption readings from a given home for some period of time (e.g, 6 months or a year), and 2) a corresponding time series with external temperature measurements, with the same granularity and for the same period of time, e.g., from a nearby weather station.

A very simple consumption profile could consist of 24 numbers: the average consumption for each hour of the day, aggregated over some or all of the input data. A simple extension is to compute two such profiles: one for weekdays and one for weekends and holidays. (We could go further and compute separate profiles for every day of the week, but, to keep the model simple and easily interpretable, we will only consider weekday-weekend splits in this paper.)

Hourly averages may reveal some high-level details about the consumption habits of a household, but we can do better. Observe that in climates with summer air conditioning usage and/or winter electric heating, a large part of the electricity consumption is temperature-sensitive; e.g., in the United States, roughly 40 percent of a home's energy consumption serves heating and cooling needs [24]. For example, Figure 1 plots the hourly consumption and external temperature (measured in degrees Celcius) for a sample home in southern Ontario, Canada, between April 2011 and October 2012 (we omit further details about the data source to preserve privacy). Observe that the peak summer consumption of this home is roughly 1.5 kilowatt-hours (kWh) higher than the peak winter consumption, which is likely due to air conditioning usage. If we could quantify the consumption of temperature-sensitive loads and remove it from the original consumption time series, the remaining consumption would give us a better idea of the occupants' routines and activities, and thus a more accurate profile.

The problem is that the relationship between consumption and external temperature is not exact, making it difficult to estimate the consumption of temperature-sensitive appliances from whole-house smart meter data. Figure 2 plots the noon-time energy consumption of the same sample home as a function of temperature; i.e., each point represents the noon-time consumption of this home on some day between April 2011 and October 2012 as well as the temperature at that time. On some days, the noon-time consumption is rel-



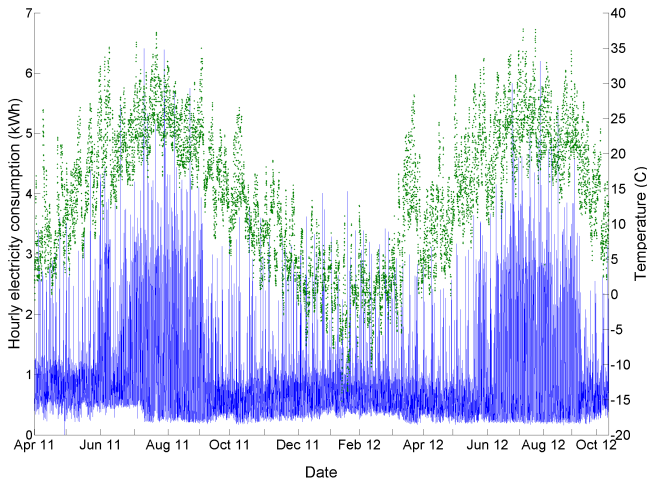


Figure 1: Hourly consumption of a sample home (blue curve with Y-axis on the left) and the temperature (green curve with Y-axis on the right), measured between April 2011 and October 2012.

atively high when the temperature is moderate or relatively low, and vice-versa. Some of these “outliers” may correspond to days when the home was empty or unusually busy. If we could remove these outliers, we could obtain a more accurate estimate of the temperature-sensitive load, and also a more accurate estimate of the remaining (routine and activity) load, which can give a more accurate and robust profile.

Our proposed solution, illustrated in Figure 3, implements the above observations. Given the smart meter and temperature time series, we will compute 48 numbers: the typical daily consumption values for each hour of the day on weekdays and weekends, after accounting for temperature-dependent load and outliers. The details of our solution are presented in the next section.

### 3. COMPUTING CONSUMPTION PROFILES

We now describe the proposed solution, beginning with an overview of the time series model that we employ (Section 3.1), followed by a discussion of how outliers and temperature effects are taken into account (Section 3.2) and how model parameters are chosen (Section 3.3). We then show how to extract consumption profiles from our time series model (Section 3.4) and we discuss several applications of the proposed profiles (Section 3.5).

#### 3.1 The PARX Model

The main idea behind our solution is to apply a time series autoregression model, specifically Periodic Auto Regression with eXogenous variables (PARX) [17]. Table 1 lists the symbols used in the remainder of the paper; in our case, we have 24 “seasons”, each corresponding to a particular hour of the day, as we will be building separate consumption models for each hour.

In general, a PARX model of order  $p$  represents a time series in terms of 1) its recent history (the most recent  $p$  data points), 2) exogenous variables, and 3) a white noise

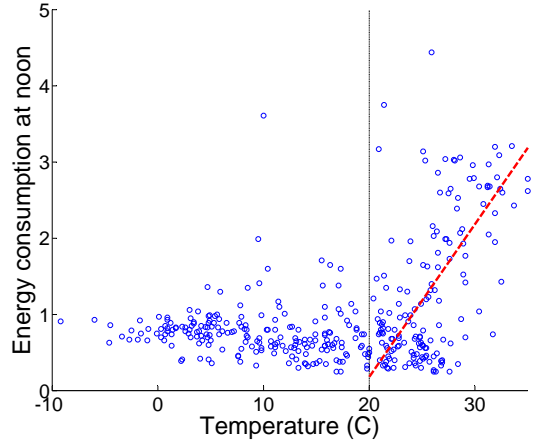


Figure 2: Hourly consumption measured at noon in a sample home versus the external temperature. The dashed line represents best linear fit for temperatures higher than 20°Celsius.

component. In our case, this can be written as

$$Y_t = \sum_{i=1}^p \phi_{i_s} Y_{t-i} + \sum_{j=1}^n \psi_{j_s} X_t^j + C_s + \epsilon_t, \quad t \in s \quad (1)$$

where  $Y_t$  is the electricity consumption at a particular hour at time  $t$ ,  $n$  is the number of exogenous variables (*i.e.*, the  $X^j$ 's),  $\epsilon_t$  is the value of the white noise component<sup>2</sup> at time  $t$ ,  $C_s$  is an intercept term, and  $s$  is the “season” index. The model parameters  $C_s$ ,  $\phi_{i_s}$ ,  $\psi_{j_s}$ , and  $\sigma_s^2$  depend on the season.

Intuitively, Equation (1) states the following. Pick a “season”, *i.e.*, some hour of the day, say, noon. The electricity consumption at noon is a linear function of the consumption at noon on the previous  $p$  days<sup>3</sup>, and of the  $n$  exogenous variables (such as temperature), plus a constant intercept term and an error term. Since each hour of the day is a separate season with its own model, the values of the coefficients  $\phi_i$  and  $\psi_j$  may be different for different hours. That is, this method is flexible enough to capture the possibility that at some hours of the day (e.g., night-time), temperature has a stronger relationship with consumption, whereas at other hours of the day (e.g., dinner-time), the load is more affected by the occupants’ activities.

#### 3.2 Exogenous Variables

As mentioned in Section 2, we want to compute the typical hourly consumption of a household, after accounting for temperature and “outliers” corresponding to periods of very low or very high consumption. This is exactly the purpose of exogenous variables. The effects of other unknown factors on the overall consumption (*i.e.*, other appliances, daily routines and patterns) will be captured in the resulting consumption profile via the auto-regressive part of the model.

<sup>2</sup>We assume that the white noise process is a sequence of independent and identically distributed random variables with zero mean and finite variance  $\sigma_s^2$ .

<sup>3</sup>More precisely, it is a function of the previous  $p$  weekdays for the weekday profile and the previous  $p$  weekends/holidays for the weekend/holiday profile.

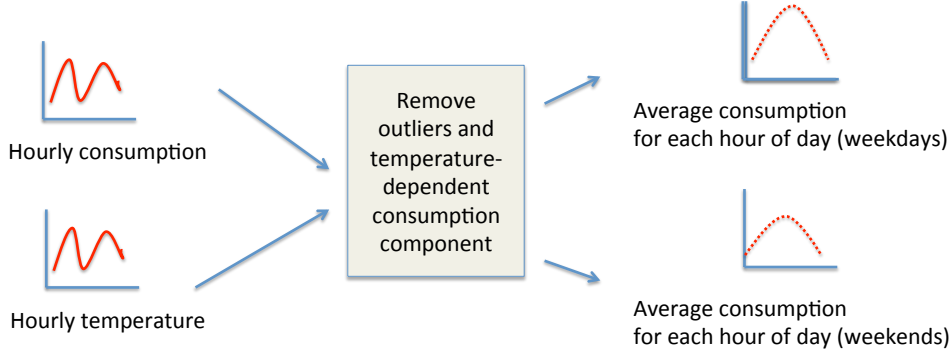


Figure 3: Overview of the proposed consumption profile approach.

$Y_t$	The original time series of household electricity consumption at time $t$
$Y_t^*$	The time series at time $t$ obtained after removing the effects of exogenous variables, representing temperature and outliers
$XT1, XT2, XT3$	Temperature related exogenous variables
$XO1, XO2$	Occupancy related exogenous variables
$\sigma_s$	The standard deviation of season $s$
$\epsilon_t$	The value of the white noise component at time $t$
$\phi_{i_s}$	The coefficient of $Y_{t-i}$ in season $s$
$\psi_{j_s}$	The coefficient of the $j^{\text{th}}$ exogenous variable in season $s$
$C_s$	The intercept term of season $s$

Table 1: List of symbols used in this paper

First, we deal with temperature. Recall Figure 2 and notice that the effect of temperature in the summer may be very different to the effect of temperature in the winter. In southern Ontario, the regression line has a positive slope at high temperatures, corresponding to the increasing intensity of air conditioning usage as temperatures climb. On the other hand, the winter effects of temperature are less pronounced, because the majority of homes, including our sample home, mainly use natural gas for heating.

The above observation implies that we cannot use a single exogenous variable to account for temperature. Instead, following previous work on modelling the effect of temperature on electricity consumption (e.g., [5, 13]), we use three variables:  $XT1$ ,  $XT2$ , and  $XT3$ . They are defined in Equations (2), (3) and (4). The coefficients of these variables represent the cooling (temperature above 20 degrees), heating (temperature below 16 degrees), and overheating (temperature below 5 degrees) slopes, respectively.

$$XT1 = \begin{cases} T - 20 & \text{if } T > 20 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$$XT2 = \begin{cases} 16 - T & \text{if } T < 16 \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

$$XT3 = \begin{cases} 5 - T & \text{if } T < 5 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Now, we show how to handle “outliers” corresponding to unusually low or high consumption. The first step is as follows. For each season (i.e., hour of the day), logically we produce a plot similar to that in Figure 2, which illustrates the relationship between temperature and consumption at that particular hour of the day across different days, using all the historical data given as input. For each value of temperature, we then compute the 10th and 90th percentiles of the consumption values. Using these values, we then define two new exogenous variables,  $XO1$  and  $XO2$ , as follows.

- At any time  $t$ ,  $XO1$  is equal to one if the consumption at  $t$  is higher than the 90th percentile of the consumption at that hour of the day and that specific temperature, as described above. It is zero otherwise.
- Similarly,  $XO2$  is equal to one if the consumption at  $t$  is less than the 10th percentile of the consumption at that hour of day and that specific temperature. It is zero otherwise.

We chose the 10th and 90th percentile values heuristically to define  $XO1$  and  $XO2$ , corresponding to very low consumption (when the home may have been empty for a long while) and very high consumption (when the household is unusually busy). We refer to  $XO1$  and  $XO2$  as occupancy-related variables.

Using all five exogenous variables, our PARX model becomes

$$Y_t = \sum_{i=1}^p \phi_{i_s} Y_{t-i} + \psi_{1_s} XT1_t + \psi_{2_s} XT2_t + \psi_{3_s} XT3_t + \psi_{4_s} XO1_t + \psi_{5_s} XO2_t + C_s + \epsilon_t, \quad \text{for } t \in s \quad (5)$$

### 3.3 Parameter Estimation

The first parameter that we need to set is  $p$ , the number of previous days to include in the auto-regressive part of the model. For each of our 24 seasons (hours of day), we tried different values of  $p$  between 1 and 48, and computed the Bayesian Information Criterion (BIC) [25]. Based on our data set of 1000 homes,  $p = 3$  gave the best results (i.e., the lowest BIC).

Once we have determined an optimal value of  $p$ , we use the standard Ordinary Least Squares (OLS) method to derive the coefficients of the PARX model for each hour of the day (repeating the process for weekdays only, and for weekends/holidays).

### 3.4 Putting it All Together

Having presented the details of our PARX framework, we are now ready to describe how the consumption profiles are derived. First, we compute our PARX model for each hour of day, separately for weekdays and weekends/holidays. We then generate a new consumption time series by taking the original values and removing the temperature-sensitive consumption as well as the outliers. For each hour of day (and separately for weekdays and weekends/holidays), we do this simply by “reversing” the model and subtracting the effects of exogenous variables. Let  $Y_t^*$  be the new consumption time series after removing temperature- and occupancy-sensitive components:

$$Y_t^* = Y_t - \psi_{1_s}XT1_t - \psi_{2_s}XT2_t - \psi_{3_s}XT3_t - \psi_{4_s}XO1_t - \psi_{5_s}XO2_t \quad \text{for } t \in s \quad (6)$$

That is, what remains in  $Y_t^*$  is just the auto-regressive part of the model.

Finally, we take the hourly averages of the corresponding  $Y_t^*$ 's, separately for weekdays and weekends/holidays, which completes the discussion of the process shown in Figure 3. Thus, our profiles consist of two vectors of 24 values, where the  $i$ th value is the typical consumption level of the given home at the  $i$ th hour of the day, after removing the effects of exogenous variables.

Figures 4 and 5 illustrate the weekday and weekend profiles of our sample home. Note that the profiles are easy to interpret and contain a great deal of useful information. For example, it is easy to see that 1) the typical hourly consumption is higher on weekdays than weekends, 2) peak weekday load occurs at 19:00 with a small peak at 9:00, while peak weekend load occurs at 17:00, and 3) the occupants of this home appear to consume more electricity between 8:00 and 11:00 on weekends than weekdays.

### 3.5 Applications

We conclude this section with a brief description of how the proposed consumption profiles can be used for two of the motivating applications listed in Section 1.

#### *Personalized recommendations for saving electricity*

Normally, we expect the hourly consumption of a typical household to decrease at night, when there is little to no activity in the home. If the consumption profile suggests that the nightly consumption of a given household remains high, then we can recommend a new refrigerator or another appliance that is always on. Note that this recommendation

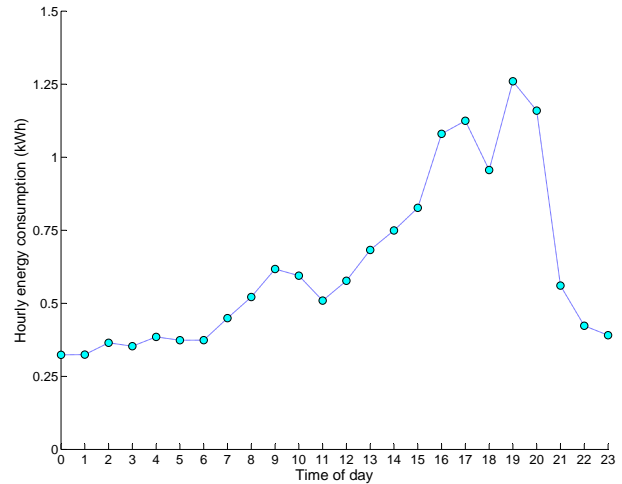


Figure 4: Weekday consumption profile of a sample home.

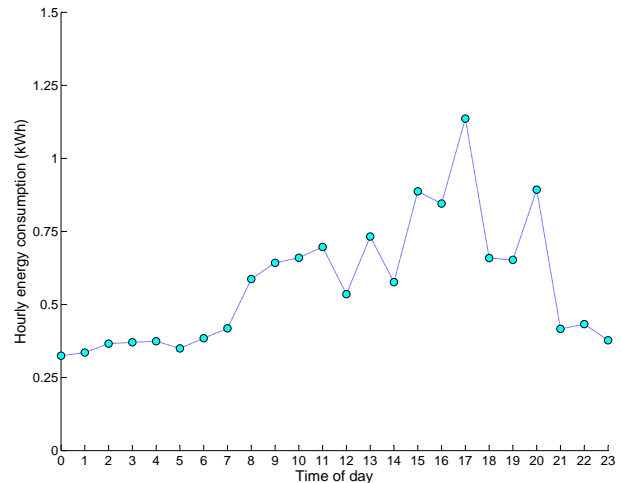


Figure 5: Weekend/holiday consumption profile of a sample home.

makes sense because we have removed temperature-sensitive load before computing the profiles. Otherwise, we would not know if the high nightly load is caused by heating and cooling or by another appliance. Similarly, if the profile shows high consumption during expensive on-peak hours regardless of temperature, then we can recommend shifting some activities (such as laundry) to off-peak hours.

Furthermore, we can generate *comparative* feedback by clustering similar consumers based on the hourly loads contained in their profiles and/or the coefficients of their exogenous variables. For instance, if a household belongs to a cluster in which other households have similar hourly loads but lower coefficients of the temperature-related exogenous variables, then we can hypothesize that this household has an inefficient air conditioning system.

## Generating realistic synthetic data as input to forecasting models or to test the scalability of smart meter data management systems

Here, the objective is to create realistic synthetic “households” based on the profiles computed from a real data set. One way to do this is as follows. First, we use a clustering algorithm such as k-means to group together similar profiles. To generate a new consumption time series, we randomly choose a cluster and use its centroid as the consumption profile of the new household. We can then choose the exogenous variable coefficients from a random member of this cluster, generate a weather forecast time series, and use this information to create the new consumption time series.

## 4. EXPERIMENTS

In this section, we evaluate the predictive power of our consumption profiles and compare it to the predictive power of representative profiling techniques from prior work—one that focuses on hourly consumption aggregates, one that uses temperature alone, and one that uses both temperature and hourly averages. We implemented the algorithms in Matlab.

Our dataset is comprised of aggregate hourly electricity consumption levels of 1000 homes from a city in southern Ontario, Canada. Measurements were taken between March 2011 and October 2012. We also obtained the ambient air temperature data of that region from the Environment Canada Website.

### 4.1 Methodology

We use two thirds of the consumption dataset of each home as the training data set for building the model, and the rest, including 170 days from April 2012 to October 2012, as the testing data set for evaluating its predictive power. For instance, to evaluate the predictive power of a model on April 1, 2012, we use consumption measurements from March 2011 to March 2012 as the training set. We extend the training set by adding days from the test set which are prior to the day for which we evaluate the predicative power. For example, April 1, 2012 is added to the training set when we evaluate predictive power for April 2, 2012.

We predict the consumption of each home using the following four profiling approaches, assuming that the hourly temperature forecast is available one day in advance.

The first is our approach, labeled *PARX*. We predict the hourly consumption on the test day, for a given hour  $h$ , as follows. First, we look up the average hourly consumption for that hour from the profile, call it  $P_h$ . We then add in the contribution of exogenous variables for that hour using the coefficients of that hour’s model. This gives us an estimate of the consumption for that hour, call it  $\hat{Y}_h$ :

$$\hat{Y}_h = Y_h^* + \psi_{1h}XT1_h + \psi_{2h}XT2_h + \psi_{3h}XT3_h + \psi_{4h}\widehat{XO1}_h + \psi_{5h}\widehat{XO2}_h \quad (7)$$

Note that in order to use our consumption profiles for predicting future consumption, we must use *estimated* values of the occupancy-related exogenous variables, denoted  $\widehat{XO1}$  and  $\widehat{XO2}$ , since we obviously do not know their true values when making the prediction. Here, we simply use the observed value of these variables in the previous hour

( $\widehat{XO1}_h = XO1_{h-1}$  and  $\widehat{XO2}_h = XO2_{h-1}$ ), although more sophisticated methods could be used to estimate these variables and further improve the predictive power of our approach.

The second approach represents methods that compute hourly aggregates from the consumption time series. In particular, we compute hourly averages over the training set and use these for prediction. We call this approach *Hourly Mean*.

The third approach represents methods that focus on the correlation between consumption and temperature [5]. This algorithm fits a three-piece linear regression model after removing very-low and very-high consumption values and uses the temperature of the test day to predict consumption. We refer to this technique as *3-Line*.

Finally, the fourth approach, proposed in [13], uses a time series model similar to PARX and also takes temperature into account (but does not account for outliers, which we do). We call this algorithm *Convergent Vector* since it computes typical hourly consumption by finding the convergent vector of the input time series.

The real value of the hourly electricity consumption on the test day is then compared with the predicted values to compute the root-mean-square error (RMSE) of each approach for each day.

### 4.2 Results

We compared the predictive power of the above four approaches using all 1000 homes in our dataset. Our findings are as follows.

- PARX outperformed Hourly Mean for 982 homes, 3-Line for 960 homes, and Convergent Vector model for 901 homes.
- The average RMSE was 0.70 for PARX, 0.81 for Hourly Mean, 0.94 for 3-Line, and 0.77 for Convergent Vector. This means that our model’s RMSE was 14 percent lower than that of Hourly Mean, 26 percent lower than that of 3-Line, and 9 percent lower than that of Convergent Vector.

Thus, although 3-Line and Convergent Vector obtained a slightly lower prediction error for a few homes, on average their prediction error is considerably higher than that of PARX. This confirms that, in most cases, incorporating historical hourly consumption, temperature dependence, and occupancy dependence results in a more representative model. Nevertheless, in some cases, temperature is highly correlated with electricity consumption, and therefore incorporating occupancy does not improve prediction accuracy. This is most likely because the bulk of the electricity consumption of these homes serves heating and cooling needs, and thus temperature alone is a very good predictor.

Figure 6 shows the average RMSE on the testing days for 20 randomly selected homes along with the RMSE values averaged over all 1000 homes on the testing days. The RMSE of PARX is lower than the RMSE of the other three approaches for all but two of these homes.

## 5. RELATED WORK

A considerable body of previous work has developed various consumption modelling techniques from household

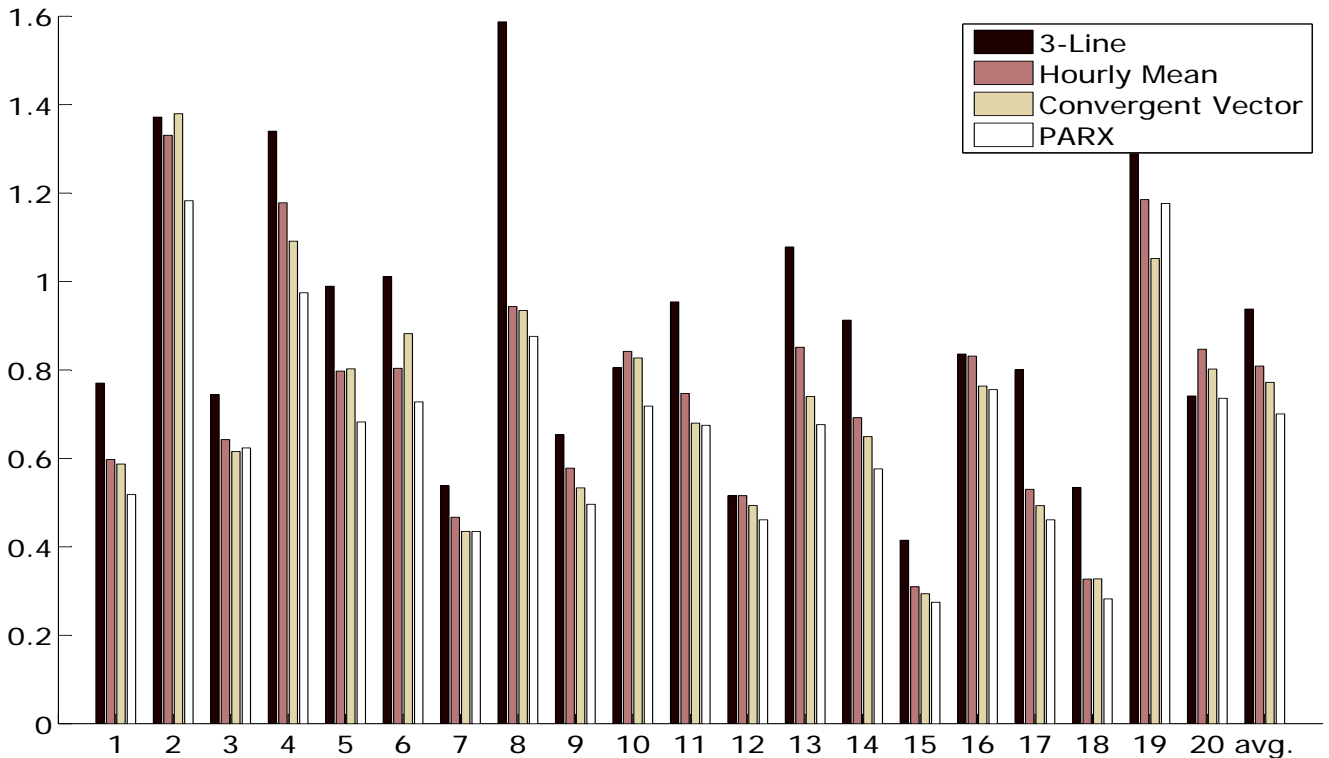


Figure 6: Average RMSE of the four approaches on 20 randomly selected homes measured on 170 testing days.

smart meter data, with applications ranging from clustering similar consumers, planning and forecasting, tariff design, electricity loss and theft detection, to providing personalized feedback on how to save electricity. Our technique may be used in many of these applications in regions where some fraction of the electricity consumption is correlated with temperature.

From a technical standpoint, previous work has approached the consumption profiling problem from two directions. One was to examine historical consumption values and compute various representative aggregates; see, e.g., [3, 8–10, 14, 20, 22]. The other direction has been to focus on the relationship between electricity consumption and temperature; see, e.g., [1, 5]. There are also techniques that combine aggregated values with temperature correlations, e.g., [13]. In this paper, our goal was to design a simple and interpretable but also accurate profiling algorithm by combining the best features of existing methods.

In particular, the two approaches most closely related to ours are by Espinoza et al. [13] and Birt et al. [5]. Our approach combines the time series modelling approach of [13] with the temperature model of [5], and additionally takes outliers into account. As we experimentally showed in Section 4, by combining and enhancing the best features of prior models, our techniques resulted in a lower prediction error.

In general, energy data management is an emerging field of study, with recent work on smart grid data management and analytics [6, 15], using Hadoop to manage smart meter data [11], imputing missing data in smart meter time series [18], and symboling representation of smart meter time series [26].

## 6. CONCLUSIONS AND OPEN PROBLEMS

In this paper, we described a simple and interpretable technique for computing electricity consumption profiles from residential smart meter data combined with temperature data. Our solution relies on auto-regressive time series modelling with exogenous variables to take into account various factors influencing electricity consumption, such as temperature and the occupants’ daily habits. Experimental evaluation using a real data set of smart meter readings from 1000 homes revealed the advantages of our method over previous work in terms of prediction accuracy.

One limitation of the proposed approach is that it is effective only for regions where some fraction of household electricity consumption is correlated with temperature, such as those with heavy air conditioning use during the summer. If this is not the case, simpler profiling techniques may be used, such as computing the average electricity consumption in each hour of the day.

We are currently building a prototype smart meter data management system, in which the proposed consumption profiling method will play a central role. We highlight several open problems in this area that we intend to study:

- Smart meter data quality: missing values are common, and unusually low or high values may indicate failing meters that need to be replaced.
- Efficient and scalable smart meter analytics: there is very little work that focuses on exploiting modern data analytics platforms, such as Hadoop, data stream engines or time series databases, for smart meter data.

- In addition to smart electricity meters, smart water meters are being introduced in many jurisdictions, including Toronto, Canada, as described at [torontowatermeterprogram.ca](http://torontowatermeterprogram.ca). This will enable large-scale water data analytics and require smart meter data management system to handle water data in addition to electricity data.

## 7. REFERENCES

- [1] A. Albert and R. Rajagopal. Building dynamic thermal profiles of energy consumption for individuals and neighborhoods. In *IEEE Big Data Conf.*, 2013
- [2] AutoGrid. [www.auto-grid.com](http://www.auto-grid.com)
- [3] C. Beckel, L. Sadamori, and S. Santini. Towards automatic classification of private households using electricity consumption data. In *4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys)*, pages 169-176, 2012.
- [4] Big Data Energy Services. [www.bigdataenergy.com](http://www.bigdataenergy.com)
- [5] B. J. Birt, G. R. Newsham, I. Beausoleil-Morrison, M. M. Armstrong, N. Saldanha, and I. H. Rowlands. Disaggregating categories of electrical energy end-use from whole-house hourly data. *Energy and Buildings*, 50(0):93-102, 2012.
- [6] M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Siksnys and T. Tusar. Data Management in the MIRABEL Smart Grid System. In *1st Workshop on Energy Data Management (EnDM)*, 2012.
- [7] C3 Energy. [www.c3energy.com](http://www.c3energy.com)
- [8] G. Chicco. Overview and performance assessment of the clustering methods for electrical load pattern grouping. *Energy*, 42(1):68-80, 2012.
- [9] G. Chicco and I.-S. Ilie. Support vector clustering of electrical load pattern data. *IEEE Trans. on Power Systems*, 24(3):1619-1628, 2009.
- [10] G. Chicco, R. Napoli, P. Postolache, M. Scutariu, and C. Toader. Customer characterization options for improving the tariff offer. *IEEE Trans. on Power Systems*, 18(1):381-387, 2003.
- [11] L. dos Santos, A. da Silva, B. Jacquin, M.-L. Picard, D. Worms, C. Bernard. Massive Smart Meter Data Storage and Processing on top of Hadoop, In *VLDB BigData Workshop*, 2012.
- [12] eSmart Systems. [www.esmart.com](http://www.esmart.com)
- [13] M. Espinoza, C. Joye, R. Belmans, and B. DeMoor. Short-term load forecasting, profile identification, and customer segmentation: A methodology based on periodic time series. *IEEE Trans. on Power Systems*, 20(3):1622-1630, 2005.
- [14] V. Figueiredo, F. Rodrigues, Z. Vale, and J. Gouveia. An electric energy consumer characterization framework based on data mining techniques. *IEEE Trans. on Power Systems*, 20(2):596-602, 2005.
- [15] U. Fischer, D. Kaulakiene, M. E. Khalefa, W. Lehner, T. B. Pedersen, L. Siksnys and C. Thomsen. Real-time Business Intelligence in the MIRABEL Smart Grid System. In *VLDB Workshop on Business Intelligence for the Real-Time Enterprise (BIRTE)*, 2012.
- [16] Green Button. [www.greenbuttondata.org](http://www.greenbuttondata.org)
- [17] H. Hurd and A. Miamee. *Periodically correlated random sequences: spectral theory and practice*, volume 355. Wiley-Interscience, 2007.
- [18] R.-S. Jeng, C.-Y. Kuo, Y.-H. Ho, M.-F. Lee, L.-W. Tseng, C.-L. Fu, P.-F. Liang, L.-J. Chen. Missing Data Handling for Meter Data Management System. In *e-Energy Conf.*, pages 275-276, 2013.
- [19] J. Miller. North american power plant air emissions. Technical Report 2-923358-11-2, Commission for Environmental Cooperation of North America, 2004.
- [20] A. Nizar and Z. Dong. Identification and detection of electricity customer behaviour irregularities. In *IEEE/PES Power Systems Conference and Exposition*, pages 1-10, 2009.
- [21] OPower. [www.opower.com](http://www.opower.com)
- [22] T. Rasanen, D. Voukantsis, H. Niska, K. Karatzas and M. Kolehmainen. Data-based method for creating electricity use load profiles using large amount of customer-specific hourly measured electricity use data. *Applied Energy*, 87(11):3538-3545, 2010.
- [23] I. Rowlands. Demand response in Ontario: exploring the issues. A report for the Independent Electricity System Operator (IESO) of Ontario, 2008.
- [24] U.S. Annual Energy Outlook 2012. [www.eia.gov/forecasts/aeo/pdf/0383\(2012\).pdf](http://www.eia.gov/forecasts/aeo/pdf/0383(2012).pdf).
- [25] W. W.-S. Wei. *Time series analysis: univariate and multivariate methods*, pages 156-157. Addison-Wesley, 2nd edition, 2006.
- [26] T. K. Wijaya, J. Eberle and K. Aberer. Symbolic representation of smart meter data. In *2nd Workshop on Energy Data Management (EnDM)*, pages 242-248, 2013.



# ECAST: A Benchmark Framework for Renewable Energy Forecasting Systems

Robert Ulbricht<sup>2</sup>, Ulrike Fischer<sup>1</sup>, Lars Kegel<sup>1</sup>, Dirk Habich<sup>1</sup>,  
Hilko Donker<sup>2</sup>, Wolfgang Lehner<sup>1</sup>

<sup>1</sup> Technische Universität Dresden, Database Technology Group, Dresden, Germany

<sup>2</sup> Robotron Datenbank-Software GmbH, Dresden, Germany

<sup>1</sup> {first name.lastname}@tu-dresden.de

<sup>2</sup> {first name.lastname}@robotron.de

## ABSTRACT

The increasing capacities of renewable energy sources and the opportunities emerging from the smart grid technology lead to new challenges for energy forecasters. Energy output fluctuates stronger compared to conventional power production. More time series data is available through the usage of sensor technology. New supply forecasting approaches are developed to better address those characteristics, but meaningful benchmarks of such solutions are rare. Conducting detailed evaluations is time-intensive and unattractive to customers as this is mostly handwork. We define and discuss requirements for efficient and reliable benchmarks of renewable energy supply forecasting tools. To cope with those requirements, we introduce the automated benchmark framework ECAST as our proposed solution. The system's capability is demonstrated on a real-world scenario comparing the performance of different prediction tools against a naive method.

## 1. INTRODUCTION

As much as for any other industry, forecasting is traditionally an important issue for utility companies. In areas like energy generation and distribution, load balancing or pricing many decisions have to be made based on uncertain data. This is the reason why beside the administration of meter data and market communication processes, the prediction of energy time series is seen as a core functionality for Energy Data Management Systems. Nowadays, with the technical challenges and opportunities emerging from the world-wide increasing capacities of renewable energy sources (RES) world-wide along with advancements like the smart grid technology, efficient and dedicated forecasting methods are being developed. Such solutions are designed to better address the typical RES characteristics like a decentralized allocation and the mainly fluctuating output owed to the changing nature of the underlying natural powers.

To cope with those challenges, a lot of research has been conducted by different communities during the past few years. However, choosing the optimal solution for a specific forecasting problem remains a formidable and intensive task for users. Despite of the large amount of available literature and both academic and practical optimization ideas, there is still a dominance of trial-and-error approaches. Results of different publications can hardly be compared, as the underlying experiments are conducted on dissimilar data sets. Also, a constant form of result evaluation is missing because different error metrics can be applied to measure output accuracy. In fact, the probability for successfully replicated results is low. Complex benchmarks tend to be time- and cost-intensive and most of the assessment procedures require expert knowledge. Integrating state-of-the-art energy supply forecasting systems into an automated benchmark framework will dramatically reduce the manual evaluation work. A suchlike composed software-supported benchmark allows for the systematical assessment and optimization of multiple tools including varying configuration settings, while saving the time of human experts. Forecasting practices in the energy sector can be improved by enabling the knowledge transfer needed to bridge the gap between scientific approaches and commercial solutions.

In this paper, we address the problem of systematic benchmarking for energy supply forecasts and introduce the *Energy Forecasting Benchmark Framework* (ECAST) as our proposed solution. The remainder of the paper is organized as follows: In Section 2 we describe the challenge of renewable energy supply forecasting. Then, we define and discuss the requirements for a dedicated benchmark against that background. In Section 3, we describe the architecture and the functional core components of our framework as well as the resulting data flows. We demonstrate the system's functionality by evaluating exemplary forecasting tools on a use case in Section 4. Finally, we conclude and outline our proposals for future developments in Section 5.

## 2. ENERGY FORECAST BENCHMARKS

Although the topic of benchmarking time series forecasting approaches seems to be a mature area covered e.g. by the M-x competition series developed by the International Insti-

tute of Forecasters, the last activities date back more than a decade and findings were obtained in a mostly domain-neutral environment [10]. Considering the background previously described in Section 1, we believe that now there is a need for benchmarks covering sophisticated energy supply forecasting solutions. Such systems were designed considering the typical characteristics of fluctuating energy production time series and the impact of external influences on the forecasting results. In this section we give a brief summary on work related to that topic and discuss the requirements for our systematical benchmarking approach.

## 2.1 Related Work

In order to make energy supply planning rational, forecasts of RES production have to be made considering weather conditions. Certainly the most influencing factors for energy output determination are the quality of the global irradiation forecast in the case of solar panels and wind speed and -direction for wind mills, respectively. Consequently, the use of precise weather forecast models is essential before reliable energy output models can be generated for such units, thus leading to the typical two-step approach presented in Figure 1. Weather forecast models can be derived using techniques like Numerical Weather Prediction (NWP), Sky Image Processing or statistical models [15]. However, this step is considered as orthogonal to our work, as grid operators and energy producers can usually purchase such data from reliable meteorological services.

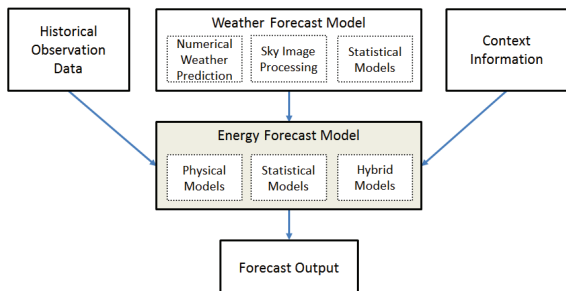


Figure 1: RES forecasting approach

As for the second step, any output obtained from the weather models is converted into electric energy output. This is done by integrating historical observation data and/or additional context information like the RES production unit’s technical details or geographical location. According to the underlying methodology, existing solutions for energy models can be classified into the categories of physical, statistical and hybrid models.

Identifying the optimal energy forecasting approach or the best-fitting software solution out of hundreds of published papers related to renewable energy supply prediction is difficult. Fortunately, there are reviews and surveys available like the work of Glassley et al. [5], who give an overview on literature for solar power forecasting but focused on irradiation prediction. A benchmark of such methods was conducted by Lorenz et al. [9] but does not cover energy models. In contrast, the work of Pedro and Coimbra [13]

assesses a couple of state-of-the-art solar energy forecasting techniques while completely excluding all exogenous inputs in their reviewed models. For wind power prediction, literature reviews are provided e.g. by Giebel et al. [4] or Monteiro et al. [11]. Another interesting approach is the *Global Energy Forecasting Competition* (GEFCom), having numerous participating research teams evaluating their models on a set of normalized wind power time series. The insights published by Hong et al. [6] show that such a competitive approach has difficulties with the simulation of real-world situations where forecasts have to be provided on a daily (or even shorter) basis. This means that newly arriving observation data is used and the forecast origin shifts with every day, thus leading to multiple time-intensive forecasting phases.

## 2.2 Benchmark Requirements

A well designed benchmark is beneficial to both system optimizing developers and evaluating customers. In this context, we observe the two vertical levels of application depicted in Figure 2: Benchmarks are commonly used to evaluate (A) a system’s overall technical performance while executing predefined tasks on different use cases or (B) the functional quality like e.g. the result accuracy of an algorithm or software implementation of interest. This can be done either (1) in a domain-neutral environment like in the case of TPC-H database benchmarks [12] or time series forecasting competitions, or (2) for a product dedicated to a specific industrial application like energy data management systems or specialized energy forecasting tools.

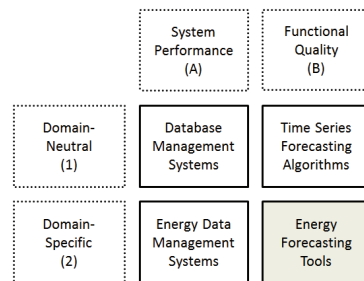


Figure 2: Benchmark design methodologies

Forecasting tools are traditionally implemented in different ways: As simple but robust spreadsheet add-ins, modules in statistical programs like R or SPSS or as dedicated stand-alone business software. Previous research in this area has shown that the latter category offers the best score for the implementation of the forecasting principles, as such software generally includes effective data preparation procedures and integrates expert knowledge for method selection support [14]. The weakness of those systems is that even by using batch versions, task automation is generally low which handicaps an efficient execution of complex benchmarks where multiple choices of conditions and parameter settings have to be tested. A more recent trend is the integration of forecasting functionality directly into database systems (e.g. [3]). This is a promising approach when considering energy supply forecasting as a massive and data



intensive process, thus requiring a higher level of automation to cope with the challenges raised from decentralized production and smart meter technology. Since this converts the forecasting tool itself into a black box thus complicating its proper adjustment and also creates dependencies on the underlying database system, we focus our work on stand-alone forecasting software.

Following the principles of time series forecasting developed by Armstrong [1], we can derive the relevant requirements for our purpose:

*Conditions.* First, the overall conditions for the experiment must be described. This includes e.g. the definition of the applied forecast horizon (static or continuous), the periods for the used original time series and the validation method to be applied on results. Possible sources of bias should be eliminated or at least described in detail if not avoidable at all.

*Data.* Usually the benchmark's underlying scenario provides the foundations for its requirements and is therefore one of the major influences for the credibility and understandability of the obtained results. To simulate a coherent business context for the target sector, the included usage models must have enough characteristics of meaningful real-world situations although it is clear and perfectly understandable that no benchmark can cover all existing use cases [16]. Applied to the energy producing sector, this means that a benchmark should include a wide range of observed energy supply time series obtained from installations allocated across different geographical regions, including all relevant and measurable external influences. Having such a use case repository allows for the easy extension of experiments to assess their generalization potential. Further, the experimental setup should match the formulated forecasting problem. This means that the underlying source data must be selected carefully considering the possible impacts on results by using real-world or synthetic or analogous data. Researchers often depend on the latter of those, as their access to real-world use cases is limited. If so, trying to find or create similar situations out of the available use cases might offer suitable alternatives. In any case access to the test data should be provided for the public (e.g. raw data for the M-competitions is always downloadable<sup>1</sup>). However, this can be problematic with real-world data sets like in the case of private energy demand and supply, because the owners will consider their data as confidential. Transformation techniques like normalization help to make the origin unrecognizable.

*Transparency.* Also, the implementation details of the evaluated methods should be disclosed in order to make sure that users understand them. This is naturally difficult when assessing commercial solutions due to the need of knowledge protection. However, identifying optimization potential for the conceptual or physical implementation layer of the system under test will be more likely if replication tests are possible. The same applies to guaranteeing both the reliability and the validity of data.

---

<sup>1</sup><http://forecasters.org/resources/time-series-data/>

*Result Evaluation.* When it comes to forecast accuracy evaluation, multiple error measures should be used to compare the obtained results as the choice of an accuracy metric can affect the ranking of the forecasting methods. The discussions frequently observable in literature show that there is no all-dominating standard accuracy evaluation criteria for time series forecasts (e.g. compare Hyndman and Koehler [7] or Chen and Yang [2]). Despite of all proposed improvements, we think that the chosen metric should be simple, easy to explain and tailored to the decision to be derived from the results. For example, the difference between over- and underestimating a wind park's expected energy output can lead to different financial penalties for its owner depending on the contractual situation.

*Limitations.* The desired benchmark is first and foremost defined as an accuracy benchmark, but anyhow under certain consideration of the calculation time which is used as a simple performance measure of the tools under test. It is definitely not meant to test the usability of the revised solutions (except parameter configuration), their result presentation quality nor every possible feature or function. We do not focus on a competitive character but want to offer systematic decision support when comparing existing systems. Other common aspects of measuring like update frequency, continuous data integration, or system reliability are considered as not being relevant for this purpose. This is why conducting an explicit cost-benefit analysis is not reasonable and excluded from our study.

### 3. SYSTEM ARCHITECTURE

In this section we describe the general architecture of our implementation. The ECAST conceptual framework is composed of four principal components as displayed in Figure 3:

1. A *Database Management System* (DBMS) as central data storage unit,
2. the *Core Logic Component* (CLC) representing a container used to encapsulate all necessary functions for system configuration, time series management, task creation and output evaluation,
3. the *Prediction-Interface* (PAPI) as connector to the forecasting systems represented by the internal and external predictors and finally
4. the *Graphical User Interface* (GUI) for necessary configurations, interactions and result presentation.

#### *Database Management System.*

The Database Management System (DBMS) represents the frameworks' central data storage unit. Its relational data structure offers tables for the purpose of storing (1) the reference parameters used for system and experiment configuration, (2) all originally observed energy- and influence time series data files which are needed for the experiments, (3) the generated forecasting tasks and (4) the obtained forecast output from the predictors. Besides the predicted time series data, the latter also includes the calculated error values and the total computation time for each experiment. For the DBMS, this results in frequent interactions in form

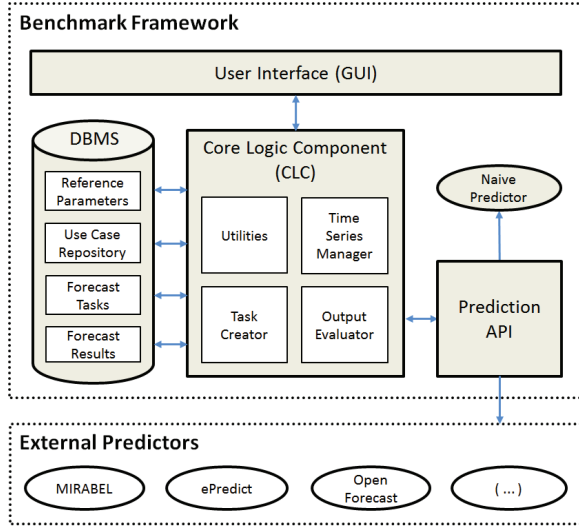


Figure 3: ECAST system architecture overview

of reading and writing operations carried out by the CLC modules Time Series Manager, Task Creator and Output Evaluator. Once source time series are stored in the DBMS, they form part of the use case repository thus easily extending the available scenarios. Data files belonging together are grouped in bundles. Additional context information like geographical location, energy type or technical installation details can be added in the use case description. That facilitates the re-identification of the stored use cases at a future date, for instance for replication tests or parameter adjustments.

### Core Logic Component.

As the name suggests, the Core Logic Component (CLC) is the heart of the framework. It contains the functionality needed to configure the system accordingly, handle input and output data for the experiments and forecasting task automation procedures. This is realized in separate modules (compare Figure 3), some of those will be described more in detail hereinafter.

*Time Series Manager.* This is responsible unit for time series data preparation and transformation. Frequently, forecasters face the problem that their source time series are too short, too noisy or having too many missing values. Overlooking the quality of source data can lead to large forecasting errors. However, we decided to reduce this functionality to input format conversion and source data validation only for the following reasons: (1) Data cleansing procedures are usually provided by Energy- or Meter Data Management Systems as this is considered being one of their core functions and (2) offering data quality improvements in the framework would bias the stand-alone performance of the forecasting tools under test, due to the fact that many of them include more or less complex data pre-processing steps as well. In order to guarantee the framework’s interoperability, all imported time series are converted into an

internal character format treating them as equidistant data structures of identical granularity throughout each scenario. This allows for an efficient storing in the use case repository and data transport to the external predictors and back, but creates a slight drawback for the human forecaster who will have to prepare the input data accordingly.

*Task Creator.* All forecast queries belonging to an experiment lead to forecast tasks. This means that the chosen settings and parameters are persisted and stored until their final execution or rather until their handover to the predictors. Depending on the experimental setup, a single forecast query can lead to multiple tasks. For example, an experiment including 2 external and the default naive predictor will lead to 3 tasks which then are sequentially executed on the same source time series. In case of predefined loops using a variable data history length for model creation or continuous forecasting horizons the number of generated tasks increases accordingly. Currently, task scheduling functionality is spared so tasks are executed immediately once the creation is completed.

*Output Evaluator.* It computes the statistical error metrics that can be applied on the output data in order to evaluate the forecast accuracy. Regarding the energy domain, the *Root Mean Square Error* (RMSE) is a recommended measure and main evaluation criterion especially for intra-day forecasts, as it addresses the likelihood of extreme values better [8]. The RMSE is found by

$$RMSE = \sqrt{\frac{\sum_{t=1}^n (P_t - P'_t)^2}{n}} \quad (1)$$

where  $P_t$  is the observed value,  $P'_t$  is the predicted value and  $n$  is the number of tuples to be compared. As the RMSE returns absolute values, we add a normalized version to allow for the comparison of the models’ performance across different scenarios thus eliminating the variance of results when including power output curves of different aggregation scales. The *Normalized Root Mean Square Error* (nRMSE) is achieved by

$$nRMSE = \frac{RMSE}{P_{max}} * 100 \quad (2)$$

with  $P_{max}$  being the maximum power output observed (only applicable if  $P_{max} > 0$ ). In the case of forecasts with day-ahead horizons or above, the mean absolute or percentage difference between observed and predicted power output can be the more appropriate evaluation criterion for users. The *Mean Absolute Error* (MAE) computes as

$$MAE = \frac{1}{n} \sum_{t=1}^n |P_t - P'_t| \quad (3)$$

while the percentage difference is expressed by the *Mean Absolute Percentage Error* (MAPE) defined as

$$MAPE = \frac{1}{n} \sum_{t=1}^n \left| \frac{P_t - P'_t}{P_t} \right| \quad (4)$$

which also implies that all tuple having  $P_t = 0$  are excluded from error calculation. As energy supply time series contain only positive values the MAPE is biased because it will favor low forecasts. Adjusted versions of MAPE are known like the *Symmetric Mean Absolute Percentage Error* (sMAPE) being one of them. Having a lower and an upper bound,

the sMAPE can provide error values between 0% and 100% which are much easier to interpret. Therefore the formula is implemented as follows

$$sMAPE = \frac{1}{n} \sum_{t=1}^n \frac{|P_t - P'_t|}{P_t + P'_t} \quad (5)$$

Another aspect of evaluation is the data range on which the error measure is applied. Commonly, all forecast values are included which leads to one returning error value based on the whole predicted time series. In addition, especially when considering the diurnal character of RES time series, also fractions of the obtained data might be interesting, for example to analyze the variance of model output accuracy on certain days. Therefore, in addition to the total error value, errors can be computed for arbitrary periods of the forecasted time series thus for example returning error time series of hourly, daily or weekly granularity.

### Prediction API

The Prediction Interface realizes the connection to each prediction tool in terms of configuration, calling the calculation method as well as the output retrieval. Several parameters are taken from the DBMS and are offered to the predictors as displayed in Figure 4: (1) The energy time series, containing the historical observation values  $P_t$  for the training and forecasting periods, (2) the influencing time series, containing the corresponding external influences to be included in the model, (3) the starting and the ending date of the training period, (4) the prediction period, indicating the start and end date of the wanted forecast and finally (5) the tool configuration, represented by a set of parameters which are passed to the respective prediction tool. With the help of those input parameters, the framework is able to externally set the configuration of the prediction tools and execute the calculations. Afterwards, the API returns the forecasted values  $P'_t$  and the total calculation time consumed by the predictor to calculate the forecast model and the forecast itself.

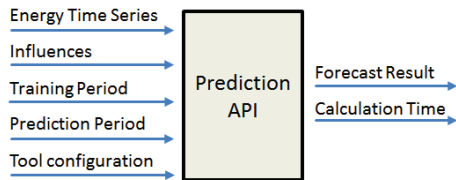


Figure 4: Prediction API methods

Due to the fact that sometimes even simple extrapolation methods may be reasonable, we include an internal *Naive Predictor* that assumes that things will not change between one day and another in a form like:

$$P'_t = P_{t-k} \quad (6)$$

with  $k$  being the number of values per day, i.e.  $k = 96$  having a granularity of 15min. Such persistence-based methods are easy to implement and commonly used to compare with the performance of more sophisticated forecasting techniques, that are represented by the external predictors connected to

ECAST. Using complex forecasting tools is worthwhile only if they are able to clearly outperform such trivial models.

### Graphical User Interface.

The user interface is designed to facilitate the experimental setup by including sophisticated functionality and triggering the internal data flows (compare Figure 5). One core function is the upload of data files into the use case repository. The external data arrives in a specified comma separated value (CSV) file, this being the lowest common form of time series data exchange and frequently seen in the energy market. Alternatively, previously stored raw time series can be selected from the use case repository. Further, the selection of tools and parameters to be assessed and the conditions needed for the generation of forecasting queries can be configured. This includes e.g. the history length of training data, forecasting horizons and loop frequencies. The setting is transformed into a XML file and later on passed to the task creator. In the post-experimental phase, the interface offers prototypical functions for output presentation like output time series plotting and error display.

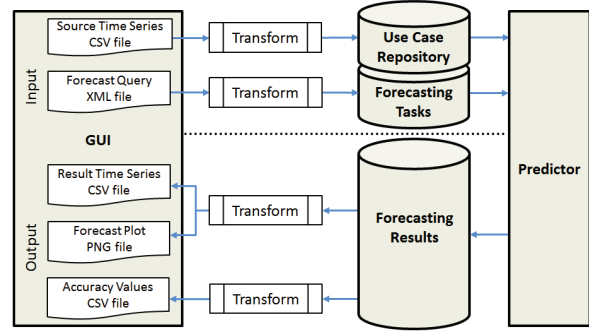


Figure 5: Logical data flow in the ECAST system

## 4. DEMONSTRATION

In order to demonstrate the functionality of the benchmark framework we conducted experiments evaluating the performance of the integrated prediction tools on two scenarios taken from the use case repository. In the following we describe the setup of the experiments and discuss the observed results.

### 4.1 Experimental Setup

The forecast quality of the external prediction tools presented in Table 1 will be compared: (1) an academic implementation originally developed for the MIRABEL project, (2) the commercial product ePredict and (3) OpenForecast, a domain-neutral open-source forecasting library. All of the chosen tools use stochastic models based on multiple regression analysis. To the best of our knowledge, none of them include relevant data pre-processing steps. Output data post-processing is reduced to the correction of negative values or completely missing as in the case of OpenForecast.

As for data, we decided to evaluate all tools on a solar- and a wind-power prediction use case. The solar power scenario

Model	Applied Algorithm	Algorithm	Data Preparation	Data Post-Processing	Source
Naive Predictor	Diurnal persistence (compare equ. 6)		No	No	-
Mirabel	Principal component analysis + Multiple linear regression		No	Negative value correction	<a href="http://www.mirabel-project.eu/">http://www.mirabel-project.eu/</a>
ePredict	Multiple non-linear regression (MARS)		No	Negative value correction, ARIMA	<a href="http://www.robotron.de/">http://www.robotron.de/</a>
OpenForecast	Multiple linear regression		No	No	<a href="http://www.stevengould.org/software/openforecast/">http://www.stevengould.org/software/openforecast/</a>

Table 1: Benchmarked forecasting tools

consists of an observed energy output time series taken from a single PV-installation located in central Germany. Data is available for the year 2012 having a resolution of 15 minutes. Corresponding influences are provided by a nearby weather station in form of hourly measurements of irradiation, outside temperature and wind speed. The usage of observed instead of forecasted influence values eliminates the prediction error naturally included in the underlying weather model thus allowing for an evaluation of the energy model performance itself. While we use the first eleven months for training, the month of December serves as prediction period. For the second scenario, a normalized wind power time series from the GEFCom 2012 wind track<sup>2</sup> was used. The installation’s location remains unknown. Historical data is available from July 2009 to December 2010 including the corresponding forecasts for wind speed and -direction, all with hourly resolution. Concurrent to the solar use case, we take all observation data except the last month for training. The forecast queries are configured with a continuous 24h-ahead horizon using a moving origin for the model. Accordingly, 31 forecasting tasks are generated for each predictor and scenario, therefore a total of 248 tasks has to be executed.

## 4.2 Benchmark results

Comparing the results for solar power prediction presented in Table 2, we can point out that all prediction tools outperform the naive benchmark in terms of RMSE, nRMSE, MAE, and MAPE. As for the sMAPE, the values for OpenForecast (0.75) and ePredict (0.70) are relatively high considering the relative position on a scale from 0 to 1. This can be explained by the impact of tuples having forecasted values  $P'_t$  close to 0 and observation values  $P_t = 0$  on the total error value. Forecasting tools optimized for solar energy can include the possibility of cutting all forecast values before dawn and after sunset (derived from geographical location) to solve such issues if properly configured. Not taken into account all tuples with  $P_t = 0$  for error calculation, the sMAPE values can be reduced to 0.32 and 0.34, respectively.

In Figure 6 the daily sMAPE values are displayed for the whole forecasting period. While the naive model has a strong fluctuation between one day and another, the external predictors show a more stable performance. Moreover, on December 12th no energy output was observed (e.g.

<sup>2</sup><http://www.kaggle.com/c/GEF2012-wind-forecasting/data>

due to snow coverage or technical failures) which explains the high error obtained from all prediction tools on that day. Figure 7 compares the measured energy output and the predicted output calculated by all predictors for December, 8th, as according to the daily error analysis good values were obtained for this period. We notice that Mirabel

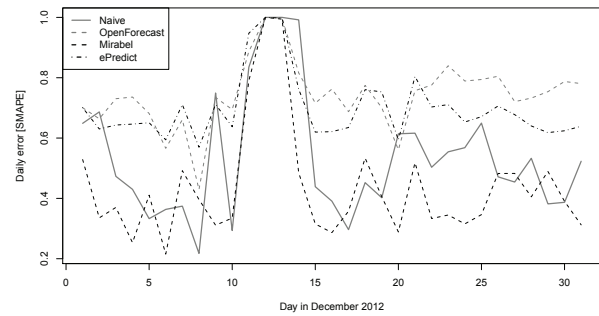


Figure 6: Daily sMAPE values for solar power prediction

and OpenForecast perform almost identical for that period, while ePredict seems to have slight advantages when capturing small peak values. It is a common drawback of using regression-based prediction models not to be able to reach peak values, as the estimations for model parameters are done by using average regression coefficients. The naive persistence method does not have that problem because data is simply copied from the previous period and accidentally energy output is very similar on both days. Also, the peak value was reached later thus leading to a shifted plot. In suchlike conditioned periods, diurnal persistence can be considered as a useful prediction method. However, it does not reach the average accuracy of the sophisticated tools using weather-aware forecasting models.

Similar results can be observed on the wind power scenario. In contrast to the solar use case, the underlying power time series has been normalized thus limiting the cross-scenario result comparison to the percental accuracy measures. Regarding the MAPE, all models show lower results than for solar power prediction. Possible explanations are higher fluctuation of wind power as there are no diurnal

Model	RMSE	nRMSE	MAE	MAPE	sMAPE	Time
Naive	5.43	11.41	1.93	1.89	0.51	<1 ms
Mirabel	3.89	8.18	<b>1.36</b>	<b>1.10</b>	<b>0.43</b>	851 ms
ePredict	<b>3.68</b>	<b>7.73</b>	1.46	1.65	0.70	999 s
OpenForecast	3.76	7.90	1.50	1.33	0.75	2389 ms

Table 2: Average forecast accuracy for 24h-ahead solar power prediction

Model	RMSE	nRMSE	MAE	MAPE	sMAPE	Time
Naive	0.27	31.48	0.20	2.91	0.53	<1 ms
Mirabel	0.21	24.44	<b>0.14</b>	<b>1.85</b>	<b>0.41</b>	511 ms
ePredict	<b>0.19</b>	<b>22.82</b>	0.16	3.18	0.44	60.8 s
OpenForecast	0.20	24.13	0.17	3.48	0.46	379 ms

Table 3: Average forecast accuracy for 24h-ahead wind power prediction

cycles and the use of weather forecasts instead of observations for model parameter estimation and forecast calculation. Those superior computation times result mainly from the smaller number of included data points, as all time series have a lower resolution and instead of 3 only 2 weather influences were used in the regression models.

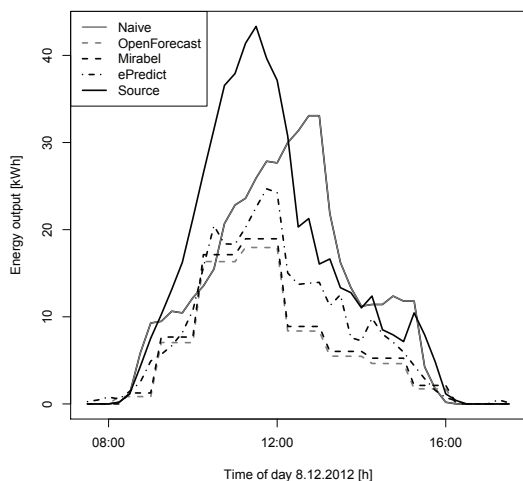


Figure 7: Model performance comparison for solar power prediction

## 5. CONCLUSIONS AND FUTURE WORK

Typical requirements for benchmarking energy forecasting tools are the definition of the overall conditions, the selection of appropriate test data and evaluation criteria and finally providing transparency. These principles were considered in the ECAST framework design: Evaluations can be conducted by configuring the desired conditions on own scenarios or given ones from the use case repository. Experimental results and initial parameter configurations are persisted in the DMBS to ease future replication attempts. Technical details of the tools under test are described as far as possible. The demonstrated use cases show that both revised energy forecasting tools really offer added value as they perform better than naive or domain-neutral methods,

although the selection of appropriate evaluation criteria influences their ranking. Basic functionality of result presentation is offered because visual inspection of plotted raw data is common and hard to replace as it helps to reveal unusual data points. Further, the efficiency of such assessments is increased by using a graphical interface for creating forecast query definitions and by substituting manual steps with automated task creation and execution.

Regarding our future work, we identified the main directions to follow: First, the Prediction API needs to be expanded as it is currently limited to statistical approaches, but physical models have to be included. They are popular especially amongst planners and investors because instead of depending on historical observation data, the production units' technical properties are used to estimate the future energy output and once they are fitted, they are accurate. We are also planning to increase the number of available external predictors, for instance, by adding for instance solutions provided by the machine learning community - variety is the key for making benchmarks more representative. Second, ECAST can be converted into decision support technology. By systematically evaluating all reasonable parametrization options, the forecasting tools will be self-adjusted to a predefined accuracy threshold. Also, combining forecasts offers additional optimization options whenever there is no solution to be found that individually outperforms in all given use cases. Using appropriate combination criteria allows for the creation of flexible hybrid models across different forecasting tools.

## Acknowledgment

The work presented in this paper was funded by the European Regional Development Fund (EFRE) under co-financing by the Free State of Saxony and Robotron Datenbank-Software GmbH. We thank the anonymous reviewers for their constructive comments that helped to improve our paper.



## 6. REFERENCES

- [1] J. Armstrong. Evaluating Forecasting Methods. In J. Armstrong, editor, *Principles of Forecasting*, volume 30 of *International Series in Operations Research & Management Science*, pages 443–472. Springer US, 2001.
- [2] Z. Chen and Y. Yang. Assessing forecast accuracy measures. Technical Report 2004-2010, Iowa State University, Department of Statistics & Statistical Laboratory, 2004.
- [3] U. Fischer, D. Kaulakiene, M. E. Khalefa, W. Lehner, T. Bach Pedersen, L. Siksnys, and C. Thomsen. Real-time Business Intelligence in the MIRABEL Smart Grid System. In *Proc. of BIRTE*, Istanbul, 2012.
- [4] G. Giebel, R. Brownsword, G. Kariniotakis, M. Denhard, and C. Draxl. The state-of-the-art in short-term prediction of wind power: A literature overview. Technical report, ANEMOS. plus, 2011.
- [5] W. Glassley, J. Kleissl, C. P. van Dam, H. Shiu, J. Huang, G. Braun, and R. Holland. Current state of the art in solar forecasting. Technical report, California Renewable Energy Collaborative (CREC), 2012.
- [6] T. Hong, P. Pinson, and S. Fan. Global Energy Forecasting Competition 2012. *International Journal of Forecasting*, 2013.
- [7] R. J. Hyndman and A. B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.
- [8] V. Kostylev and A. Pavlovski. Solar Power Forecasting Performance—Towards Industry Standards. In *1st Int. Workshop on the Integration of Solar Power into Power Systems*, Aarhus, Denmark, 2011.
- [9] E. Lorenz, J. Remund, S. C. Müller, W. Traunmüller, G. Steinmaurer, D. Pozo, J. A. Ruiz-Arias, V. L. Fanego, L. Ramirez, M. G. Romeo, and Others. Benchmarking of different approaches to forecast solar irradiance. In *24th European Photovoltaic Solar Energy Conference*, pages 1–10. Hamburg, Germany, 2009.
- [10] S. Makridakis and M. Hibon. The M3-Competition: results, conclusions and implications. *International Journal of Forecasting*, 16:451–476, 2000.
- [11] C. Monteiro, R. Bessa, V. Miranda, A. Botterud, J. Wang, G. Conzelmann, and Others. Wind power forecasting: state-of-the-art 2009. Technical report, Argonne National Laboratory (ANL), 2009.
- [12] R. Nambiar, M. Poess, A. Masland, H. R. Taheri, M. Emmerton, F. Carman, and M. Majdalany. TPC Benchmark Roadmap 2012. In *Selected Topics in Performance Evaluation and Benchmarking*, pages 1–20. Springer Berlin Heidelberg, 2013.
- [13] H. T. C. Pedro and C. F. M. Coimbra. Assessment of forecasting techniques for solar power production with no exogenous inputs. *Solar Energy*, 86(7):2017–2028, 2012.
- [14] L. Tashman and J. Hoover. Diffusion of Forecasting Principles through Software. In J. Armstrong, editor, *Principles of Forecasting*, volume 30 of *International Series in Operations Research & Management Science*, pages 651–676. Springer US, 2001.
- [15] R. Ulbricht, U. Fischer, W. Lehner, and H. Donker. First Steps Towards a Systematical Optimized Strategy for Solar Energy Supply Forecasting. In *Proc. of the Joint ECML/PKDD 2013 Workshops*, 2013.
- [16] L. Wyatt, B. Caufield, and D. Pot. Principles for an ETL Benchmark. In *TCPCTC 2009, LNCS 5895*, pages 183–198. Springer Berlin Heidelberg, 2009.

# Energy Data Management: Where Are We Headed? (panel)

Torben Bach Pedersen  
Aalborg University  
tbp@cs.aau.dk

## ABSTRACT

This panel paper aims at initiating discussion at the Third International Workshop on Energy Data Management (EnDM 2014) about the important research topics and challenges within Energy Data Management. The author is the panel organizer, extra panelists will be recruited from the workshop audience.

## Keywords

Energy Data Management, Architectures, Information Models

## 1. QUESTIONS AND CHALLENGES

The panel should try to answer (at least) the following questions:

- What was already done within energy data management, and what is still missing?
- What are the scientific challenges?
- What are the technical challenges?
- What are the challenges that necessitate an interdisciplinary approach?

Below, some of the panel organizer's personal opinions on these topics are listed.

One thing that is still missing is a broad range of open benchmark datasets that can be used to develop robust and effective methods for various energy data management tasks, e.g., datasets that provide detailed measurements of device usage and energy consumption at a fine-grained level for a larger number of households.

Several scientific challenges are still open, including a) the development of robust and effective methods and techniques for prediction of energy production and consumption down to the device level; b) the development of methods capable of extracting and predicting flexibilities in energy usage;

c) the development of scalable techniques for aggregating, scheduling, and disaggregating micro-level flexibilities, e.g., in individual device consumptions, to large-scale macro-level units suitable for balancing energy supply and demand at the higher levels;

On the technical level, there is still a lack of community-wide agreed-upon common definitions of data and information concepts, e.g., standardized ontologies specifying common concepts. Also, the standardization of communication protocols, e.g., for communicating available flexibilities, is very important.

Interdisciplinary challenges are perhaps the hardest to meet, and include the interplay between computer scientists developing scalable techniques for energy data management, human-computer interaction designers exploring how and at which level of detail to interact with a smart grid system, e.g., in the home, and economists developing new business and energy taxation schemes that can ensure the (financial) interest of all the many involved parties (consumer, producers, distributors, traders, balance responsible parties, etc.) while still generating a tax revenue at the same level as current schemes. An example of these disciplines continuously interacting to develop viable solutions for the truly smart grid is found in the Danish Totalflex project [www.totalflex.dk](http://www.totalflex.dk).

## 2. PANEL/WORKSHOP ORGANIZER

**Prof. Torben Bach Pedersen** is full professor of computer science at Aalborg University, Denmark. He received his Ph.D. in 2000. His research interests span Big Data and business intelligence topics such as data warehousing, multidimensional databases, OLAP, and data mining, with a focus on non-traditional and complex types of data. He has published more than 140 peer-reviewed papers on these topics. He has served as PC Chair for DaWaK 2009+10, DOLAP 2010, and SSDBM 2014, General Chair for SSTD 2009, and on numerous program committees, including SIGMOD, (P)VLDB, ICDE, and EDBT. He has worked on energy data management since 2007, was involved in the MIRABEL EU FP7 project on energy data management, as is now leading the research in the large interdisciplinary Danish project, TotalFlex.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

# Exploratory Search in Databases and the Web (ExploreDB)

Georgia Koutrika (HP Labs, USA)

Laks V.S. Lakshmanan (University of British Columbia, Canada)

Mirek Riedewald (College of Computer and Information Science, USA)

Kostas Stefanidis (University of Crete & FORTH-ICS, Greece)



# Exploratory Search in Databases and the Web

Georgia Koutrika  
HP Labs, Palo Alto  
koutrika@hp.com

Laks V.S. Lakshmanan  
Department of Computer  
Science, University of British  
Columbia  
laks@cs.ubc.ca

Mirel Riedewald  
College of Computer and  
Information Science,  
Northeastern University  
mirek@ccs.neu.edu

Kostas Stefanidis  
ICS-FORTH, Heraklion  
kstef@ics.forth.gr

## Introduction

The traditional way a user interacts with a database system is through queries. Structured query languages, such as SQL for relational data, XQuery for XML, and SPARQL for RDF data, allow users to submit queries that may precisely capture their information needs, but users need to be familiar with the underlying ontology and data structure and of course the query language itself. Moreover, users need to some extent be familiar with the content of the database and have a clear understanding of their information needs. These requirements stand as the weaknesses of this interaction mode. As data stored in databases grows in unprecedented rates and becomes accessible to diverse and less technically oriented audience, new forms of data exploration and interaction become increasingly more attractive.

The World Wide Web represents the largest and arguably the most complex repository of content. Users seek information on the web through two predominant modes: by browsing or by searching. In the first mode, the interaction between the user and the data repository is driven directly by the user's interpretation of their information need and their information foraging constraints. In the latter mode, a search engine typically mediates the user-data interactions and the process starts with the user entering query-terms that act as surrogates for the user information goals. Free-text queries allow end-users a simple way to express their information needs independently from the underlying data model and structure, as well as from a specific query language. Given a query, the most common strategy has been to present the results as a ranked list. Users have to subsequently peruse the list to satisfy their information needs through browsing the links and/or by issuing further queries.

However, the information in the web gets rapidly diversified both in terms of its complexity as well as in terms of the media through which the information is encoded, spanning from large amounts of unstructured and semi-structured data to semantically rich available knowledge. Increasing de-

mands for sophisticated discovery capabilities are now being imposed by numerous applications in various domains such as social media, healthcare, telecommunication, e-commerce and web analytics, business intelligence, and cyber-security. Yet, many of these data are hidden behind barriers of language constraints, data heterogeneity, ambiguity, and the lack of proper query interfaces.

Furthermore, the complexity and heterogeneity of the information implies that the associated semantics is often user-dependent and emergent. Individual aspects like age, gender, profession or experience are often not taken into account, for example the difference in searching between children and adults. In addition, most common systems still assume that the user has a static information need, which remains unchanged during the seeking process. Hence, they are strongly optimized for lookup searches, expecting that the user is only interested in facts and not in complex problem solving.

Consequently, there is a need to develop novel paradigms for exploratory user-data interactions that emphasize user context and interactivity with the goal of facilitating exploration, interpretation, retrieval, and assimilation of information. A huge number of applications need an exploratory form of querying. Ranked retrieval techniques for relational databases, XML, RDF and graph databases, text and multimedia databases, scientific and statistical databases, social networks and many others, is a first step towards this direction. Recently, several new aspects for exploratory search, such as preferences, diversity, novelty and surprise, are gaining increasing importance. From a different perspective, recommendation applications tend to anticipate user needs by automatically suggesting the information which is most appropriate to the users and their current context. Also, a new line of research in the area of exploratory search is fueled by the growth of online social interactions within social networks and web communities. Many useful facts about entities (e.g. people, locations, organizations, products) and their relationships can be found in a multitude of semi-structured and structured data sources such as Wikipedia<sup>1</sup>, Linked Data cloud<sup>2</sup>, Freebase<sup>3</sup>, and many others. Therefore, novel discovery methods are required to provide highly expressive discovery capabilities over large amounts of entity-relationship data, which are yet intuitive for end-users.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

<sup>1</sup><http://wikipedia.org>

<sup>2</sup><http://linkeddata.org>

<sup>3</sup><http://freebase.com>

## The ExploreDB Workshop

The purpose of the ExploreDB workshop is to bring together researchers and practitioners that approach data exploration from different angles, ranging from data management, information retrieval to data visualization and human computer interaction, in order to study the emerging needs and objectives for data exploration as well as the challenges and problems that need to be tackled.

In this first workshop instance, we have put together a program, comprising a keynote talk, six research papers, and a panel, that examines data exploration from the standpoints of data visualization, information retrieval, web search, data mining, and database queries. We are grateful to all the authors who submitted papers. We would also like to thank Daniel Keim for accepting to be the keynote speaker, and our reviewers who did their best in delivering thorough reviews on time.

## The ExploreDB Organization

### The workshop co-chairs

Georgia Koutrika  
*HP Labs, Palo Alto, USA*

Laks V.S. Lakshmanan  
*Department of Computer Science,  
University of British Columbia, Canada*

Mirek Riedewald  
*College of Computer and Information Science,  
Northeastern University, USA*

Kostas Stefanidis  
*Institute of Computer Science, FORTH, Hellas*

### Program committee

Ira Assent, Aarhus University, Denmark

Wolf-Tilo Balke, TU Braunschweig, Germany

Carlos Castillo, Qatar Computing Research Institute, Qatar

Nicola Ferro, University of Padua, Italy

Minos Garofalakis, Technical University of Crete, Greece

Melanie Herschel, University of Paris South, France

H. V. Jagadish, University of Michigan, Ann Arbor, USA

Haridimos Kondyakis, ICS-FORTH, Greece

Mohamed Mokbel, University of Minnesota, Minneapolis, USA

Kjetil Kjetil Nørnvåg, Norwegian University of Science and Technology, Norway

Evaggelia Pitoura, University of Ioannina, Greece

Julia Stoyanovich, Drexel University, Philadelphia, USA

Letizia Tanca, Politecnico di Milano, Italy

Martin Theobald, University of Antwerp, Belgium

Panayiotis Tsaparas, University of Ioannina, Greece

Yannis Tzitzikas, University of Crete and ICS-FORTH, Greece

Jun Yang, Duke University, Durham, USA

Demetris Zeinalipour, University of Cyprus, Cyprus

# Exploring Big Data using Visual Analytics

Daniel A. Keim  
Department of Computer and Information Science, University of Konstanz  
Daniel.Keim@uni-konstanz.de

## Abstract

Never before in history data is generated and collected at such high volumes as it is today. For the exploration of large data sets to be effective, it is important to include the human in the data exploration process and combine the flexibility, creativity, and general knowledge of the human with the enormous storage capacity and the computational power of today's computers. Visual Analytics helps to deal with the flood of information by integrating the human in the data analysis process, applying its perceptual abilities to the large data sets. Presenting data in an interactive, graphical form often fosters new insights, encouraging the formation and validation of new hypotheses for better problem-solving and gaining deeper domain knowledge. Visual analytics techniques have proven to be of high value in exploratory data analysis. They are especially powerful for the first steps of the data exploration process, namely understanding the data and generating hypotheses about the data, but they also significantly contribute to the actual knowledge discovery by guiding the search using visual feedback.

In putting visual analysis to work on big data, it is not obvious what can be done by automated analysis and what should be done by interactive visual methods. In dealing with massive data, the use of automated methods is mandatory - and for some problems it may be sufficient to only use fully automated analysis methods, but there is also a wide range of problems where the use of interactive visual methods is necessary. The presentation discusses when it is useful to combine visualization and analytics techniques and it will also discuss the options how to combine techniques from both areas. Examples from a wide range of application areas illustrate the benefits of visual analytics techniques.

## Short Bio

DANIEL A. KEIM is professor and head of the Information Visualization and Data Analysis Research Group in the Computer Science Department of the University of Konstanz, Germany. He has been actively involved in data analysis and information visualization research for about 20 years and developed a number of novel visual analysis techniques for very large data sets. He has been program co-chair of the IEEE InfoVis and IEEE VAST as well as the ACM SIGKDD conference, and he is member of the IEEE VAST as well as EuroVis steering committees. Dr. Keim got his Ph.D. and habilitation degrees in computer science from the University of Munich. Before joining the University of Konstanz, Dr. Keim was associate professor at the University of Halle, Germany and Senior Technology Consultant at AT&T Shannon Research Labs, NJ, USA.

# On the Suitability of Skyline Queries for Data Exploration

Sean Chester, Michael L. Mortensen, and Ira Assent  
Data-Intensive Systems, Aarhus Universitet  
Åbogade 34 8200-Århus N, Denmark  
{schester, illio, ira}@cs.au.dk

## ABSTRACT

The skyline operator has been studied in database research for multi-criteria decision making. Until now the focus has been on the efficiency or accuracy of single queries. In practice, however, users are increasingly confronted with unknown data collections, where precise query formulation proves difficult. Instead, users explore the data in a sequence of incrementally changing queries to the data to match their understanding of the data and task. In this work, we study the skyline operator as a tool in such exploratory querying both analytically and empirically. We show how its results evolve as users modify their queries, and suggest using our findings to guide users in formulating reasonable queries.

## 1. INTRODUCTION

Say you have never been to America and you find yourself in Manhattan searching for a restaurant. Where do you even begin? Probably, you want something close, but quite what is “close” may not be clear. If you might go to a show later, several locations can be equally valid reference points for “close.” Perhaps you prefer something inexpensive, but having never been to Manhattan, what really is “expensive”? Search sites can help, but only if you know for what to look.

The *skyline* operator is said to be useful in this context, because it identifies the data points (restaurants) that express the best trade-offs between the dimensions of interest (proximity, rating, and price). But what if the user wants to *explore* the data, and may evolve new preferences throughout the process? He/she may decide that price, after all, is no concern, or not to look at any more pizzerias. For the skyline to be useful in this interactive process, it is crucial that one can continually add constraints and change dimensions of interest without completely changing the results that he/she sees. If the skyline filters too many points that it did not filter before, the user will likely be as mystified as the users in the skyline user study of Magnani et al. [8].

An interactive skyline has been assumed in several contexts (e.g., skycube computation [5], dynamic skylines [4], visualization [8], anytime computation [9], and preference elicitation [1, 7]), but how the interaction affects the skyline

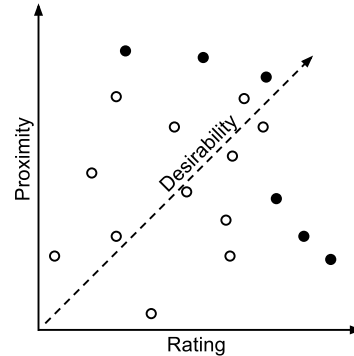


Figure 1: Example of a skyline. The black points are in the skyline because no other points have, relative to them, both a higher rating and proximity.

is not well understood. If a small query changes produce radically different skylines, which is theoretically possible (Section 3), then an interactive skyline would not make sense.

Nevertheless, if the skyline accommodates making incremental changes to a query formulation, it has potential to help an exploratory user. So, in this paper, we take a first look at how suitable the skyline is when repeatedly executed on slightly different views of the data. In particular, we ask:

1. What can theoretically happen to the results of a skyline when a query is incrementally vs. arbitrarily reformulated (Section 3)?
2. Does being in one skyline make being in a skyline for similar queries more likely? Or is one query’s result uninteresting once the user’s preferences evolve?
3. How often are the theoretical effects in Section 3 empirically observed in real and synthetic data (Section 4)?

## 2. BACKGROUND

Figure 1 illustrates the skyline [2], a filtration tool. Given a set  $\mathcal{D}$  of  $n$  points  $p = (p_0, \dots, p_{d-1})$  in  $d$  dimensions, the *skyline* consists of all points  $p \in \mathcal{D}$  that are not dominated by any other points  $q \in \mathcal{D}$ . A point  $q$  dominates another point  $p$  if  $\forall 0 \leq i < d, q_i \geq p_i$  and  $\exists 0 \leq i < d, q_i > p_i$ .<sup>1</sup> That is to say, for any pair of non-equal points, if one is in the skyline it must have a higher value than the other on

<sup>1</sup>The assumption of preferring larger values is WLOG: one can multiply any attribute by  $-1$  in preprocessing.

some attribute. The skyline consists of all points that are not inferior to some other point.

However, the number of skyline points can be quite large [3] and a user may find only some dimensions and values to be interesting; so, the skyline operator should be combined with subspace projections (then called a *subspace skyline* [12]) and with range constraints (then called a *constrained skyline* [10]). Our general form of a skyline query is then:

```
SELECT <subspace>
FROM <tables>
WHERE <range constraints>
SKYLINE <min/max specifications>
```

A user specifies dimensions are of interest (the subspace), min and max values for those dimensions (range constraints), and whether he/she prefers smaller or larger values on each attribute (the specifications). To (logically) execute such a query, one first applies the constraints and projections, and then computes the skyline of the resultant, filtered dataset.

In an exploratory context, the skyline is not the terminus of the process. After observing the results, the user will reformulate the query to match evolved understanding *in an incremental manner*. That is to say, subsequent queries in an exploratory process are not disjoint, because if they were, that would imply that the user is completely dissatisfied with the results of the first search, since he/she deliberately excluded them from the second search. This is effectively restarting. We will focus on largely overlapping queries, which suggest some successful interactivity.

More precisely, we define an *incremental change* as additions *or* removals of subspace dimensions or an edit to one range constraint. Such incremental changes can produce query results similar to the ones before. Any larger changes can be decomposed into a sequence of incremental ones. However, while the points satisfying the constraints likely are similar after an *incremental change*, the extent to which the skyline changes is not well known.

## Problem statement

Given a *baseline* query, consisting of a subspace projection and a set of range constraints, and an *incremental change* to that query, how many points do the skyline of the baseline and the skyline of the modified query have in common?

## 3. THEORETICAL EFFECTS

In this section, we look at what *can* happen to the results of a skyline query after an incremental change.

### 3.1 Effect of varying constraints

Consider a skyline query applied to some baseline constraints (Figure 2). The results are very specific to the constraints posed; for example, although points  $p$  and  $l$  would be part of the skyline if there were no constraints, neither match any user constraints. On the other hand, whereas  $g$  and  $h$  are not part of the unconstrained skyline, they become skyline points if  $l$  is eliminated by the constraints.

In fact, *every* point can be part of the skyline for *some* set of constraints and *no* point is guaranteed to always be a skyline point. Therefore, an arbitrary change in constraints can have unpredictable consequences to the skyline: possibly adding new skyline points, removing existing ones, or both.

What we show here, however, is that if a user makes only an *incremental change*, the behaviour is predictable. There

are four *effects* that can be observed, two types of skyline point addition and two types of skyline point removal:

1. **Addition (A)**: A point only satisfies the new constraints and so becomes a skyline point;
2. **Removal (R)**: An existing skyline point only satisfies the old constraints and so is removed from the skyline;
3. **Promotion (P)**: A point becomes a new skyline point because all the points that dominated it are removed by the new constraints;
4. **Demotion (D)**: An existing skyline point is removed from the skyline because it becomes dominated by some point that only satisfies the new constraints, but not the old ones.

Although an arbitrary change to constraints can induce any or all effects, an incremental change cannot. On a given attribute, there can be both an upper (U) and a lower (L) constraint, either of which can be increased (I) or decreased (D). We analyze each of these four cases:

**LD**. In Figure 2(b), the lower constraint is decreased. This **adds**  $a$  to the skyline, which has a high  $y$ -value but had an  $x$ -value outside the constraints. An LD change to dimension  $D$  can add points to the skyline if they are in the subspace skyline on the remaining dimensions, but it can never **remove** or **promote** points. New points have lower  $D$ -values than, and so could never dominate, existing skyline points.

**LI**. In Figure 2(c), the lower constraint is increased. This **removes**  $d$  from the skyline: it no longer matches the constraints. LI changes to dimension  $D$  consider no new points (so cannot **add**) and remove those smallest on  $D$ . Their removal cannot result in a **promotion**, because points satisfying the new constraints must have a higher  $D$ -value so cannot have been previously dominated by the removed points.

**UD**. In Figure 2(d), the upper constraint is decreased. The points  $n$  and  $k$  are **removed**, but point  $i$ , which was dominated by  $k$ , is **promoted**. The **add** effect cannot occur, because all points matching the new constraints matched the old constraints. Therefore, for a point  $q$  to become a new skyline point subject to the new constraints, the point that dominated  $q$  in the old constraints must be eliminated.

**UI**. In Figure 2(e), the upper constraint is increased. Consequently, points  $l$  and  $m$  are **added** and points  $\{g, h, k, n\}$  are all **demoted** because they are dominated by  $l$ . Existing skyline points cannot be simply **removed**, because they necessarily match the new constraints: they can only be eliminated from the skyline by becoming dominated.

A user can control which of the effects could happen by making an incremental change (one constraint), e.g., LU to decrease the skyline size. Composing constraint modifications on different attributes is only predictable if the modifications are all of the same type. *How much* each of these effects is observed we evaluate empirically (Section 4).

### 3.2 Effect of varying subspace projections

Incremental changes to subspace projections can also create four effects. As with the constrained skylines, there are two types of skyline addition and two types of removal:

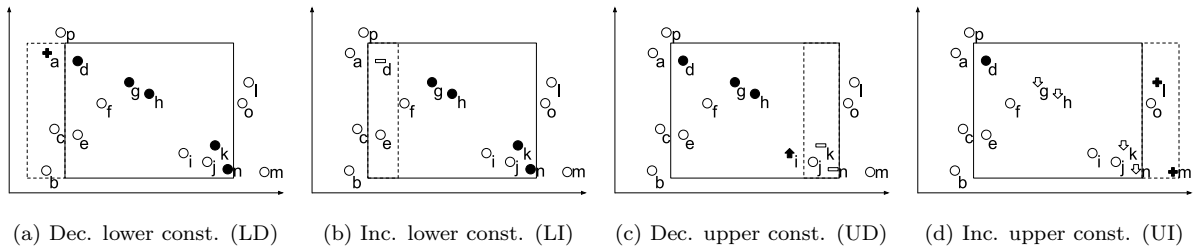


Figure 2: The four incremental constraint changes. The solid rectangle shows the baseline constraints and the dotted lines indicate the modification to the constraints. Solid points are in the skyline and hollow points are not. A *plus* indicates an **addition**; a *minus*, a **removal**; an *upwards arrow*, a **promotion**; and a *downwards arrow*, a **demotion**.

1. **Addition (A)**: A point dominated in the old subspace becomes *incomparable* to the points that dominated it, and thus becomes a skyline point in the new subspace;
2. **Removal (R)**: A skyline point in the old subspace is now *dominated* in the new subspace, so no longer belongs to the skyline;
3. **Homogenization (H)**: A point not in the skyline in the old subspace becomes a skyline point because it is *identical* to some skyline point in the new subspace;
4. **Differentiation (D)**: A skyline point that *was identical* to another skyline point in the old subspace is dominated by that skyline point in the new subspace.

For an example, consider the dataset,  $\mathcal{D}_{\text{ex}}$ , below, and the incremental addition of subspace dimensions, from  $\{x_0\}$  to  $\{x_0, x_1\}$  to  $\{x_0, x_1, x_2\}$  (i.e., just the first value of each point, then the first two, then all three).

$$\mathcal{D}_{\text{ex}} = \begin{cases} p = (1, 2, 2) \\ q = (1, 1, 2) \\ r = (0, 2, 3) \end{cases}$$

To begin, the skyline in  $x_0$  is  $\{p, q\}$ , since  $p$  and  $q$  both have higher values than  $r$  on  $x_0$ , but not than each other. By adding dimension  $x_1$ , the skyline becomes  $\{p\}$ , an instance of **differentiation**. Point  $q$  is removed from the skyline because it is no longer identical to point  $p$ , instead now dominated by it. Finally, adding the last dimension, the skyline becomes  $\{p, r\}$ , an instance of **addition**. Point  $r$  is added to the skyline because it has a higher value than the other skyline point,  $p$ , on  $x_2$ ; so, they have become incomparable.

In the other direction, we observe the inverse effects, first the **removal** of  $r$  and then then **homogenization** of  $q$ .

So, whether one adds or removes a dimension, points can be both added or removed from the skyline. Therefore, theoretically at least, one cannot anticipate how the skyline might change going from one subspace to a neighbour without using sophisticated preprocessing techniques, such as those in [11]. So, we will determine the actual *frequency* of these effects empirically (Section 4).

#### A note about Distinct Value Condition

Effects H and D create unpredictability when changing subspaces. Thus the motivation for Distinct Value Condition [11], which ensures monotonicity. In particular, if no value appears twice in the dataset for the same attribute, then a point in the skyline for some subspace will always remain in the skyline after adding any number of other dimensions.

## 4. EMPIRICAL INVESTIGATION

In the previous section, we investigated what theoretically happens to the result of a skyline query if one makes incremental changes to the subspace or constraints. In this section, we investigate *how often* each of these effects empirically occur. Our strategy with these experiments is to execute an initial subspace or constrained query, modify the query formulation by a variable extent, and measure the occurrences for each effect defined in Section 3.

### 4.1 Setup

To observe incremental changes, we conduct one suite of experiments in which we adjust constraints (Section 4.2.1) and one in which we add dimensions to a subspace (Section 4.2.2). We briefly describe implementation details (Section 4.1.1), the datasets that we use (Section 4.1.2), and the methodology (Section 4.1.3).

#### 4.1.1 Implementation Details

We first apply the constraints/subspace projections onto the data with a short *awk* program, and then apply a known skyline algorithm. When applying constraints, we use the state-of-the-art skyline algorithm, *BSkyTree* [6] (implemented by the original authors). As the *BSkyTree* algorithm does not handle duplicate points—which occur quite frequently in some subspaces—we use our own implementation of *BNL* [2] when applying subspace projections.

#### 4.1.2 Datasets

For the experiments with constraints, we primarily use the standard skyline synthetic data generator [2],<sup>2</sup> with parameters we discuss in Section 4.1.3. The synthetic data permits drawing general conclusions with respect to the specific data distributions. To also observe behaviour on real data, we choose the *nba*<sup>3</sup> dataset, a standard benchmark for skyline research, consisting of statistics for 21961 basketball player-seasons. We use eight of the statistics, *gp*, *pts*, *asts*, *pf*, *fga*, *fgm*, *fta*, and *ftm*, because others contain frequent NULLs.

The behaviour between subspaces is only interesting without Distinct Value Condition (Section 3.2); so, we again use the *nba* dataset, which has duplicate values, but not the synthetic data, which does not. We add the *automobiles* dataset,<sup>4</sup> which has 406 points and 8 dimensions (although we only use the first seven, because *origin* is non-ordered).

<sup>2</sup><http://http://pgfoundry.org/projects/randdataset>

<sup>3</sup><http://www.databasebasketball.com>

<sup>4</sup><http://stat-computing.org/dataexpo/1983.html>

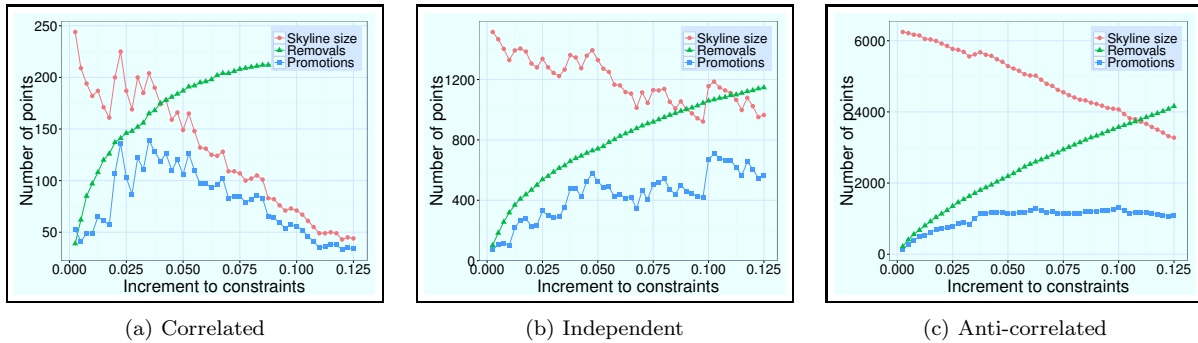


Figure 3: Decreasing an upper constraint -  $UD$ . ( $n = 100K$ ,  $d = 6$ )

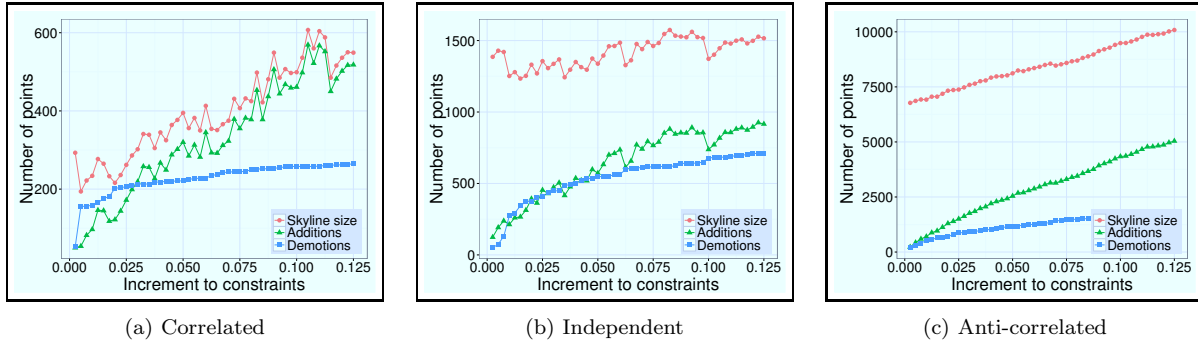


Figure 4: Increasing an upper constraint -  $UI$ . ( $n = 100K$ ,  $d = 6$ )

We choose this dataset to observe behaviour with many duplicate maximum values: 27% of the cars have the maximum (8) of cylinders, 7.5% of the cars have the maximum (1982) year, and 4.5% of the cars have the maximum (98.0) displacement. Non-maximal values are duplicated, too (e.g., 6 cylinders). All datasets are normalized to the range  $[0, 1]$  to make the interpretation of our plots easier.

### 4.1.3 Methodology

There are many ways to reformulate a query, so many variables to consider empirically. We focus on those most natural for a user to tune and containing polar cases that demonstrate the range of skyline behaviour.

Studying constraints, we use synthetic data, introducing more variables but more generalizable findings. We hold the number of (and ergo density of) points constant at  $100K$  and dimensionality at 6. As typical in skyline literature, we vary the distribution (correlated, independent, and anti-correlated), since the skyline size varies with correlation.

We pose a constant initial seed constraint that prunes 75% of the data ( $\geq 0.75$  or  $\leq 0.25$ ). This is a reasonable first constraint, equivalent to asking only for restaurants with ratings of at least 4.0 (on a range of 1 to 5). We then compare the skyline result to that for constraints in 50 increments of 0.0025. We vary the direction of these changes (as per Section 3), both increasing and decreasing the constraint. We place the initial constraint in two different, extreme locations: one in the maximal direction (a lower constraint) and one in the minimal direction (an upper constraint). We do this for every dimension. In total, this produces 72 combinations on synthetic data, each for which we plot the 50 comparisons between the baseline and the modified constraint.

We do the same with the *nba* dataset to produce 48 combinations (not varying correlation, using all 8 attributes).

For the subspace investigation, there are fewer variables. As discussed above, we use only real datasets. We iterate each of the  $(2^d - 2)$  proper, non-empty subspaces and, for each, compare the skyline result to that produced after adding 1 or 2 dimensions. We only *add* dimensions, because removal is symmetric. This produces 6050 and 1932 combinations for the *nba* and *automobiles* datasets, respectively.

## 4.2 Discussion

In this section, we describe the salient observations on skyline behaviour when adjusting constraints (Section 4.2.1) and adding dimensions to subspaces (Section 4.2.2).

### 4.2.1 Effects of constraints

We present here our findings on the effects of incremental constraint changes. We present, first, for upper constraint changes ( $UD$  and  $UI$ , in the terminology of Section 3.1), and, second, for lower constraint changes ( $LD$  and  $LI$ ).

**Upper constraints.** Figures 3a-3c show the skyline size, along with the number of **removals** and **promotions** for a representative  $UD$  case on synthetic datasets. We do not show **additions** nor **demotions**, since, in agreement with Section 3.1, there are none.

In all distributions, we see that as the size of change increases, the **removals** increase steadily and the **promotions** vary throughout. This is especially apparent for correlated data, where about 80% of the original skyline is removed after a 0.065 decrease of the upper constraint and about 80 **promotions** occur after a slight change of 0.020.

This corresponds to a user, say, decreasing his/her budget for a restaurant, so the large and variable number of promotions implies he/she will see plenty new options when filtering out that which is most expensive. We also see the skyline size has a net decrease; so, as the user would expect, narrower ranges produce fewer results. Consequently, a *UD* change is appropriate for a user who wants to see *new* points without substantially changed the baseline constraints.

Figures 4a-4c show the skyline size, **additions** and **demotions** for the *UI* case, where a user is, say, increasing his/her budget. The overall trend is an increasing skyline size with only slight drops on account of **demotions**. Again, the correlated data shows large changes to the skyline even on very slight changes to constraints: minute changes induce over 100 **demotions**, creating an immediate local drop in skyline size. To an exploratory user most of the original results will seemingly be no longer valid.

Overall, on upper constraint modifications, we see dramatic changes for correlated data because the skyline will generally be located around the upper boundary of the data space for all of the correlated dimensions. Thus when one upper constraint is changed, it is likely to affect every skyline point and the change is more immediate than we see for other distributions. For all distributions, we see that small changes to constraints can yield significant changes to the skyline result. Assuming rational users want to predict the outcome of their input actions, a viable strategy for interactively using the skyline is to only make especially small changes to upper constraints. If changes are too large, the skyline may seem unpredictable and uncontrollable.

**Lower constraints.** We omit plots for the **additions** caused by an *LD* change and for the **removals** induced by an *LI* change for space. In agreement with Section 4.2.1, we only observe one type of effect for each of these changes and any **additions** or **removals** have a proportionate effect on the skyline size. The effect grows linearly with the size of the change in constraints. A user adjusting a lower constraint is probably trying to filter the results, because it is contrary to the direction of his/her preferences. The results confirm that this is a viable strategy: lowering (raising) the constraint increases (limits) the output.

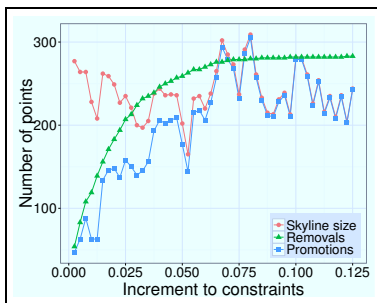


Figure 5: NBA - *UD*. ( $n = 21961$ ,  $d = 8$ )

**Real data.** To investigate the effects from constraints on real data, we conducted the same experiments for the *nba* dataset. Figures 5 and 6 show the effects of decreasing the upper constraint on the *fgm* attribute and of increasing the upper constraint on the *fta* attribute, respectively. As was

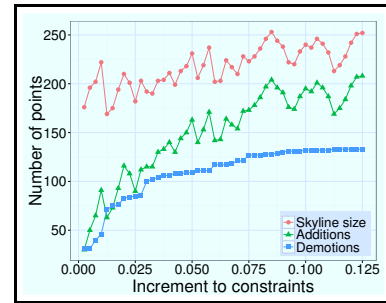


Figure 6: NBA - *UI*. ( $n = 21961$ ,  $d = 8$ )

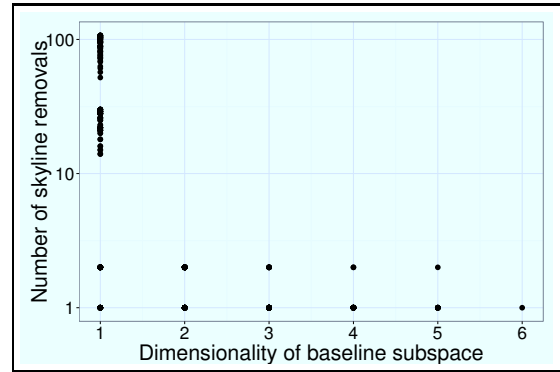


Figure 7: Scatterplot of skyline removals compared to dimensionality of the baseline subspace - *Automobiles*

the case with the synthetic data, we see a steady stream of **removals** and **demotions**. What is novel here is that while most of the skyline has been replaced after only a 0.04 decrease of the upper constraint in Figure 5 the skyline neither increases nor decreases heavily. The same trend is visible in Figure 6, where the skyline size has minimal variance.

This trend is interesting for an interactive analysis, since it shows a dataset like *nba* provides different skylines of comparable sizes, depending on the user’s needs and expressed conveniently in the constraints. This further supports the strategy of using *UD* and *UI* cases to explore different skyline points in an incremental manner.

Results for the *LD* and *LI* cases are omitted due to space constraints, but confirm the trends shown in the synthetic experiments, supporting the strategy of using the *LD* and *LI* changes to regulate the skyline size.

#### 4.2.2 Adding dimensions to subspace projections

In these experiments, we add dimensions to baseline subspace projections to observe how often the effects analyzed in Section 3.2 empirically occur. We only add dimensions, because the behaviour is symmetric (can be interpreted by reading the plots right-to-left) when removing dimensions. To a user, **Differentiations** (and **homogenizations**) are counter-intuitive, because they request more (less) data and then obtain fewer (more) results. So, we investigate under what circumstances this will impact the exploratory process.

On the *nba* dataset, we never observe **differentiations** for any configuration of experiment parameters. This is an interesting result because it shows that even in the presence of many duplicate values, one may not see any **differentiations** if those duplicates do not occur on the maximum



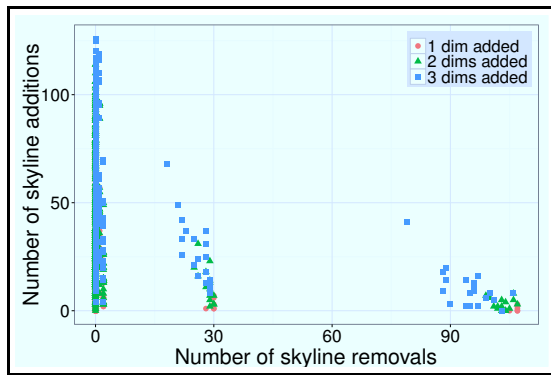


Figure 8: Scatterplot of skyline additions and removals caused by adding to baseline subspace - *Automobiles*

values. For the *nba* dataset, for example, there is seldom a tie for a record statistic such as *most points scored in a single NBA season*. So it is not surprising that *competitive* players do not have identical statistics in subspaces. Exploring the subspaces in this dataset will be quite intuitive.

On the *automobiles* dataset, we observe **differentiations**. Figure 7 shows their frequency with respect to the the baseline subspace dimensionality when one dimension is added. Even with all dimensions, **differentiations** occur (points exist, albeit only 1). Beyond one dimension, all cases include the *cylinders* attribute, which has the most duplicated maximum values. On other attributes, there are no **differentiations** if the baseline has at least two dimensions. This illustrates that while **differentiations** are rare when the baseline projection is on several dimensions, they do occur, and should be illustrated to the user so the results appear stable.

The second plot, Figure 8, shows how common are simultaneous **additions** and **differentiation**. The simultaneity is less desirable for exploring data, because it makes it harder to contrast subsequent queries. We see that if only one dimension is added (the triangular points), the effect can be predominantly **additions** or predominantly **differentiations** (on the axes), but not both (in the middle). This is because **differentiations** pre-suppose a high degree of homogeneity on maximal values, only one of which needs a high value on the new dimension in order to continue dominating all the points that are not in the baseline skyline. As the number of dimensions added goes up, there is a trend towards more mixed effects, because homogeneous points need to continue dominating non-baseline-skyline points over more new dimensions. By adding 3 dimensions, it is quite common to see roughly equal **additions** and **differentiations**; then, the result size has not changed (and thus is not easier to interpret), but the mixture of points has (which is counter-intuitive).

In summary, **differentiations (homogenizations)** are uncommon when adding (removing) one dimension, especially if starting with several dimensions. They are unlikely to occur at all if the duplicated values in the dataset are not on the maximal values. Nevertheless, they occur, even in high dimensional subspaces, and need to be illustrated to an exploratory user who would otherwise be confused.

## 5. CONCLUSION

In this work, we investigated how the skyline performs

as a tool for exploratory data analysis. We analyzed how the skyline is affected by incremental changes with standard database operators (projection and selection) by defining the theoretical effects that one *can* observe, and measuring the frequency with which these effects *are* observed empirically.

A central motivation for the skyline is to disencumber the user from having to specify query parameters, and this research helps advance that objective. We envision that query recommendation can benefit from understanding the effects that incremental changes will have. If the goal is to produce new results, one can suggest some *UD/UI* changes or a dimension to remove. If the goal is to control the output size, *UD/UI* changes or additional dimensions can be automatically recommended. Future work can investigate strategies for producing these recommendations. Learning the consequences of manipulating query parameters is only a first step in exploring the expansive possibilities for how users and skyline-based systems can interact and in guiding exploration of new data in a principled manner.

## 6. ACKNOWLEDGEMENTS

This work has been supported in part by the Danish Council for Strategic Research, grant 10-092316.

## 7. REFERENCES

- [1] W.-T. Balke, U. Güntzer, and C. Lofi. Eliciting matters – controlling skyline sizes by incremental integration of user preferences. In *DASFAA*, pages 551–562, 2007.
- [2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [3] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.
- [4] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang. A safe zone based approach for monitoring moving skyline queries. In *EDBT*, pages 275–286, 2013.
- [5] J. Lee and S. Hwang. Qskycube: Efficient skycube computation using point-based space partitioning. *PVLDB*, 4(3):185–196, 2010.
- [6] J. Lee and S. Hwang. Scalable skyline computation using a balanced pivot selection technique. *Information Systems*, 39:1–21, January 2014.
- [7] J. Lee, G.-w. You, S. Hwang, J. Selke, and W.-T. Balke. Interactive skyline queries. *Information Sciences*, 211:18–35, 2012.
- [8] M. Magnani, I. Assent, K. Hornbæk, M. R. Jakobsen, and K. F. Larsen. Skyview: a user evaluation of the skyline operator. In *CIKM*, pages 2249–2254, 2013.
- [9] M. Magnani, I. Assent, and M. L. Mortensen. Anytime skyline query processing for interactive systems. In *DBRank*, 2012. No. 7.
- [10] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1):41–82, 2005.
- [11] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang. Towards multidimensional subspace skyline analysis. *TODS*, 31(4):1335–1381, 2006.
- [12] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, page 65, 2006.

# Hippalus: Preference-enriched Faceted Exploration

Panagiotis Papadakos and Yannis Tzitzikas  
Institute of Computer Science, FORTH-ICS, GREECE, and  
Computer Science Department, University of Crete, GREECE  
{papadako|tzitzik}@ics.forth.gr

## ABSTRACT

In this work we describe and evaluate **Hippalus**, a system that offers exploratory search enriched with *preferences*. **Hippalus** supports the very popular interaction model of Faceted and Dynamic Taxonomies (FDT), enriched with user actions which allow the users to express their *preferences*. The underlying preference framework allows expressing preferences over attributes (facets), whose values can be hierarchically valued and/or multi-valued, and offers automatic conflict resolution. To evaluate the system we conducted a user study with a number of tasks related to a “car selection” scenario. The results of the comparative evaluation, with and without the preference actions, were impressive: with the preference-enriched FDT, all users completed all the tasks successfully in 1/3 of the time, performing 1/3 of the actions compared to the plain FDT. Moreover all users (either plain or expert) preferred the preference enriched interface. The benefits are also evident through various other metrics.

## 1. INTRODUCTION

Users access large amounts of information resources (documents or data) mainly through search functions, where the user types a few words and the system (web search engine, query engine) returns a linear list of hits. While this is often satisfactory for focalized search, it does not provide enough support for recall-oriented (exploratory) information needs. As several user studies have shown, a high percentage of search tasks are *exploratory* ([1]), the user does not know accurately his information need (e.g. in WSE users provide in average 2.4 words [4]), and such needs cannot be satisfied by a single ‘hit’.

A highly prevalent model for exploratory search is the interaction of Faceted and Dynamic Taxonomies (FDT), which allows the user to get an overview of the information space (e.g. search results) and offers him various groupings of the results (based on their attributes, metadata, or other dynamically mined information). These groupings enable the user to restrict his focus gradually and in

a simple way (through clicks, i.e. without having to formulate queries), enabling him to locate resources that would be difficult to locate otherwise (especially the low ranked ones). This model is currently used in various domains: e-commerce (e.g. eBay), booking applications (e.g. booking.com), library and bibliographic portals (e.g. ACM Digital Library), museum portals like Europeana, mobile phone browsers, and many others.

The enrichment of search mechanisms with preferences could be proved useful for recall-oriented information needs, because such needs involve decision making. However the current approaches for preference-based access [13], mainly from the area of databases, seem to ignore that users should be acquainted with the information space and the available choices for describing effectively their preferences. On the other hand, the available personalization services over FDT, do not allow the explicit expression of preferences, but try to automatically suggest the most preferred facets and values according to a number of different criteria. In this way the user somehow “loses” the control of the interaction.

In this work, we describe and evaluate **Hippalus**, a preference enriched FDT system, for exploratory browsing. Its functionality is founded on the preference framework described in [15], whose distinctive features is the ability to express preferences over attributes whose values can be hierarchically organized, and/or multi-valued, while scope-based rules resolve automatically the conflicts. We conducted a user study of the **Hippalus** system, over a number of tasks, with and without the preference actions. The gathered results were impressive. Even though the available choices were few (50 cars), with the preference-enriched FDT all users completed all the tasks successfully in 1/3 of the time, performing 1/3 of the actions compared to the plain FDT. Moreover all of the users (either plain or expert) preferred the preference enriched interface.

## 2. BACKGROUND & RELATED WORK

### Faceted and Dynamic Taxonomies

Modern environments should guide users in exploring the information space and in expressing their information needs in a *progressive* manner. Systems supporting FDT offer a simple, efficient and effective way for exploratory tasks [12]. *Dynamic taxonomies* (faceted or not) is an interaction framework based on a multi-dimensional classification of may heterogeneous data objects allowing users to browse and explore the information space in a guided, yet unconstrained way through a simple visual interface. Features of this framework include: (a) display of current results in

multiple categorization schemes (called facets - or just attributes), (b) display of facets and values leading to non-empty results only, (c) display of the count information for each value (i.e. the number of results the user will get by selecting that value), and (d) the user can refine his focus gradually, i.e. it is a session-based interaction paradigm in contrast to the stateless query-and-response dialogue of most search systems. Moreover, and as shown in [10, 7], this interaction paradigm can act complementarily to the traditional query-and-response dialogue, by post-processing and post-exploring the results returned by a classical search system.

In any case, the user explores or navigates the information space (either the entire information base, or the search results), by setting and changing his *focus*. The notion of focus can be *intensional* or *extensional*. Specifically, any conjunction of values (or any boolean expression of values in general) is a possible *focus*. For example, the initial focus can be the empty, or the top term of a facet. However, the user can also start from an arbitrary set of objects, e.g. the search results returned by a common WSE. In that case we can say that the focus is defined *extensionally*.

### FDT and Preferences

Most FDT systems output facets and zoom-points in lexicographical order, or order facets and zoom-points based on the number of indexed documents. Other systems, like eBay, only present a manually chosen subset of facets to the users, and the zoom-points are again ranked based on the number of indexed documents.

Recently, various approaches try to automatically present the most “useful” facets and zoom-points according to various criteria like *set-cover ranking* of indexed objects [2], *interestingness* over a number of criteria [3], or use *collaborative* [8] and *content* filtering [14] to rank facet-values pairs. Minimum-effort driven navigational techniques for enterprise databases, that rapidly drill down to the most prominent tuples are described in [11]. In the same manner, but for zoom-points, [5] propose a system for faceted navigation using a cost model of user navigation. A browsing-oriented approach for facet ranking and grouping of facets and their values according to different intuitions and metrics is provided in [16]. There are various other works, discussed in [9]. But none of them allows users to explicitly express their preferences during the exploration process.

To the best of our knowledge the only model that allows users to define explicitly the desired preference structure in a *gradual* and *flexible* manner, also exploiting attributes with *hierarchically organized values* and possibly *set-valued*, is the one proposed in [15]. In this paper we describe and evaluate the Hippalus system, which supports the above framework.

## 3. THE HIPPALUS SYSTEM

Hippalus is a publicly accessible web system<sup>1</sup> demonstrating a preference-enriched FDT-based exploratory process. It offers actions that allows the user to order facets, values, and objects using *best*, *worst*, *prefer to* actions (i.e. relative preferences), *around to* actions (over a specific value), or actions that order them lexicographically, or according to their values or count number. Furthermore, the user is able to compose object related preference actions, using *Priority*,

<sup>1</sup><http://www.ics.forth.gr/isl/Hippalus>

*Pareto*, *Pareto Optimal* (i.e. skyline), and *Combination* (i.e. order according to priority; the rest actions are the least prioritized and use *Pareto* composition) compositions. All the above functionality is offered in an efficient way, by using the algorithms described in [15].

The information base that feeds Hippalus is represented in RDF/S (using a schema adequate for representing objects described according to dimensions with hierarchically organized values). For loading and querying such information Hippalus uses Jena<sup>2</sup>, a Java framework for building Semantic Web applications. Hippalus offers a web interface for FDT exploration, enriched with the aforementioned preference actions through HTML 5 *context menus*<sup>3</sup>. The performed actions are internally translated to statements of the preference language described in [15], and are then sent to the server through HTTP requests. The server parses the preference statements and if they are valid, computes the respective preference bucket order. Finally, the resulting according to preference ranked list of facets, terms or objects is sent to the user’s browser.

### 3.1 Interaction and User Interface Design

The most widely adopted approach or policy for FDT visualization (evidenced by the UI design of global systems like booking.com, eBay), is to use a left bar for the facets and the corresponding zoom points. This is also the case for the Hippalus system (Figure 1.a<sup>4</sup>). Hippalus displays the preference ranked list of objects in the central part of the screen, while the right part is occupied by information that relates to the information thinning process (object restrictions), preference actions history and preference composition. It offers the preference related action through right-click activated pop-up menus (through HTML5 context menus). This policy does not require allocating permanent screen space for these actions. However the user should be aware that these options exist. The design of the preference actions, includes actions that are anchored to one element, and this makes the right-click activated actions straightforward. Moreover, the proposed preference-based framework supports also actions that concern two elements, i.e. relative preferences like *Korean*  $\succ$  *European*. Figure 1(c) shows how such statements can be expressed through a context menu: the action is anchored to *Korean* and the available menu options guide the user through the options that are valid in this specific situation and the specific user focus. Notice the icon in the *European* option in the right most menu. By pressing it, the preference is recorded.

At any time the user can restrict his focus to any hard constraint (i.e. the information thinning process), and his soft constraints (i.e. expressed preferences) will be applied to the current restriction of the object set.

Finally, since the number of objects can be very large, the user can specify a threshold, so that preferences are applied only when the number of objects is reduced under this threshold<sup>5</sup>. Options and parameters regarding the system functionality can be set through a drop-down menu (i.e. simple or full support of preference menus, threshold, etc.)

<sup>2</sup><http://jena.apache.org/>

<sup>3</sup>Available only to firefox 8 and up.

<sup>4</sup>In the following screens, the underlying information base contains data about 50 cars, as described in Section 4.

<sup>5</sup>The user can reduce the number of objects by selecting facets and zoom-points, restricting his focus.

Regarding the description of the current state, the user is able to view not only the intentional description of his current state, but also the accumulated preferences that he has formulated. Finally, the user is able to store and load his preferences, since exploration is a time depth process.

### 3.2 Interaction Example

Here we describe a more complete scenario demonstrating how hard and soft constraints can be specified by the user, in an easy and gradual manner. It also aims at making clear the merits of the underlying preference framework (preference inheritance and scope-based conflict resolution). A video showcasing this scenario is available online<sup>6</sup>.

The first screen (Fig. 1.a) shows the 50 cars and the left bar shows the attributes, their values (which can be hierarchically organized), accompanied by the number of their occurrences. Figure 1.b shows that one can expand broad values, like **Asian** (from the attribute **Manufacturer**), and that by clicking on the value **Korean** the focus is restricted on three Korean cars. Notice that the left bar has been updated, i.e. only the values that appear in the restricted set are presented (all attributes have count up to 3). With additional clicks the user can further reduce the focus, e.g. from the attribute **Fuel Type** we can see that one of the cars consumes **Diesel** and two cars **Gasoline**. By clicking on **Gasoline** we see these two cars and by mouse over one of them the user gets its “Object Card” showing all attributes of that car. At the right bottom frame the user can see the history of his clicks and can undo any click.

Preferences are activated through right-click menus. Suppose we cancel all clicks and assume that we want to express that we prefer Korean cars to European. This means that we do not want to see only Korean; we just want to get them ranked higher than European. This is shown in Figure 2.a(top) where we see that now the user is getting a linear list of blocks of equally preferred objects, here the first contains Korean cars, the next one European (thanks to inheritance the user does not have to say anything about German, Italian, French, etc).

It is important that preferences can be expressed incrementally and at any point during the interaction. For example suppose that in addition the user prefers prices around 12,000. He can use the action **around 12,090** as shown in Figure 2.a(bottom). We can see that the object order now becomes more refined (the figure shows 14 blocks). Notice that the first block contains one Korean (**Hyundai**) and one **Fiat**. This happens because both of his preference actions have the same priority (and Fiat is closer to 12090). If the user wants to give higher priority to one preference he can use preferences composition tool at the right frame. Figure 2(b) shows the object order obtained after expressing that the preferences over manufacturers have higher priority than the preferences over prices.

At any time the user can click on a value from a facet to restrict the current focus, which is now a preference-based list of cars. For instance, if the user wants to see only cars having two doors, he can click on 2 in the attribute **Doors**. We can see that now he gets only 8 cars, which are ranked according to his preferences so far. The user could cancel this extra restriction from the object restriction history.

In general the user can combine object restriction (or relaxation) actions and preference actions in any order.

<sup>6</sup><http://www.youtube.com/watch?v=Cah-z7KmlXc>

## 4. EVALUATION

The objective was to investigate whether even in a small dataset (50 cars), the addition of preferences to FDT would make the users more effective and satisfied, without making the interaction complex to use or learn. To this end we compared two different UIs: a) **Hippalus** system with exploration and browsing capabilities only, where preference functionality was disabled ( $UI_1$ ) and b) **Hippalus** system with exploration and browsing capabilities and preference functionality enabled ( $UI_2$ ). Regarding  $UI_2$ , we configured **Hippalus** to provide only preference actions affecting objects (i.e. users were not able to express preferences regarding attributes and their values).

### Information Base

We used an information base of 50 cars, where each car is described by 23 attributes, as shown in Fig. 1.a. A number of attributes have hierarchically organized values like **Manufacturer** and **Drive\_System**, while the rest like **Doors**, **Year**, **Price**, etc. are flat.

### Tasks

We created two variations of equal tasks for the plain and two equal variations for the expert users of the evaluation. In our context task equality is defined as tasks that consist of the same kind of preference actions and criteria. For each task, the first subtask was designed around prioritized composition of preference actions, while the second one over *Pareto* composition. Plain users tasks used only 3 criteria, while the expert ones were more difficult and complicated, using a total of 6 different criteria. The tasks that users completed are available in [9, Chapter 6].

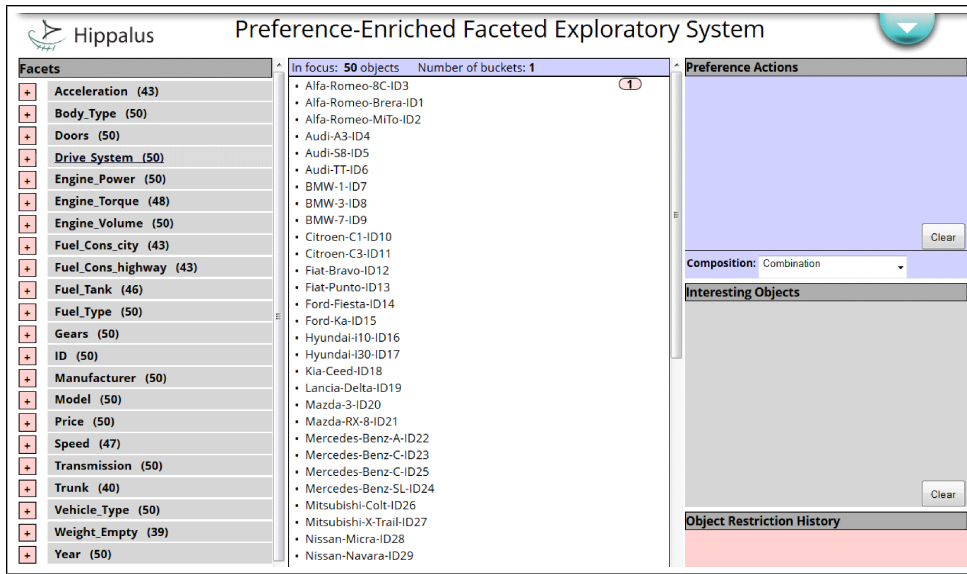
### Participants

26 males and females of varying age (i.e. between 23-43 years) and expertise (i.e. tertiary education - PhD level) participated in this study. We formed two groups. The first group, named *plain* users, consisted of 20 regular users, while the second one, *expert* users, consisted of 6 people with a prior experience in using multi-dimensional browsing and access systems that support preferences. Before starting the evaluation, users were given a simple tutorial of 15 minutes. In more details, initially users were given a description of the information base (domain, attributes). In the next five minutes they were described the interactive process of information thinning and finally the rest of the tutorial demonstrated the preference actions by showing specific examples. Finally, users were allowed to get acquainted with the UI and complete a number of simple tasks.

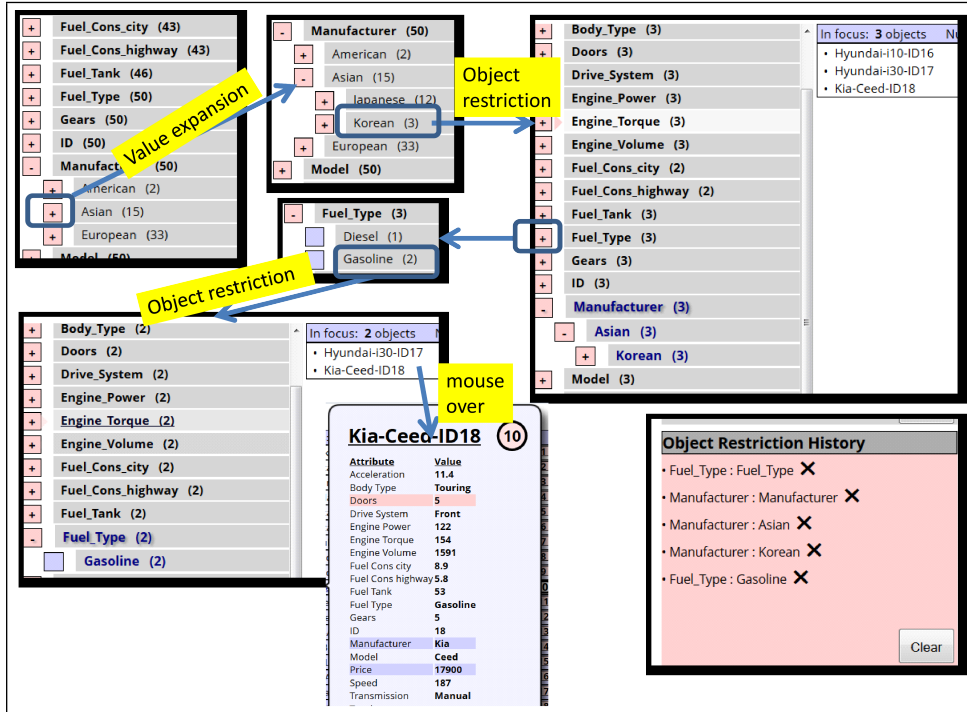
### Evaluation

The users were asked to evaluate  $UI_1$  and  $UI_2$  using the previously described tasks. Regarding  $UI_1$ , users completed the tasks by using the available information thinning functionality to restrict their focus and by inspecting the available cars. For  $UI_2$ , on top of the information thinning functionality they could also submit preference actions. For both UIs, the users provided the set of cars which they believed fulfilled the needs of each task, by drag-&-dropping cars into the “Interesting Objects” frame in the right middle part of the system (Fig. 1.a).

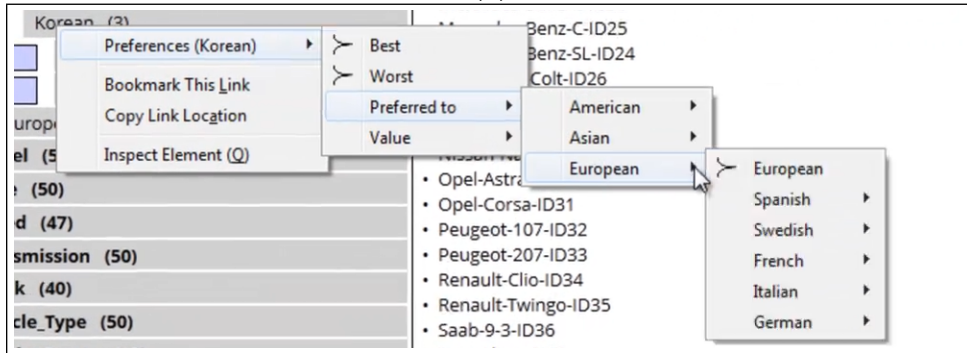
In order to control for order effects and to increase the chance that results can be attributed to the experimental



(a)

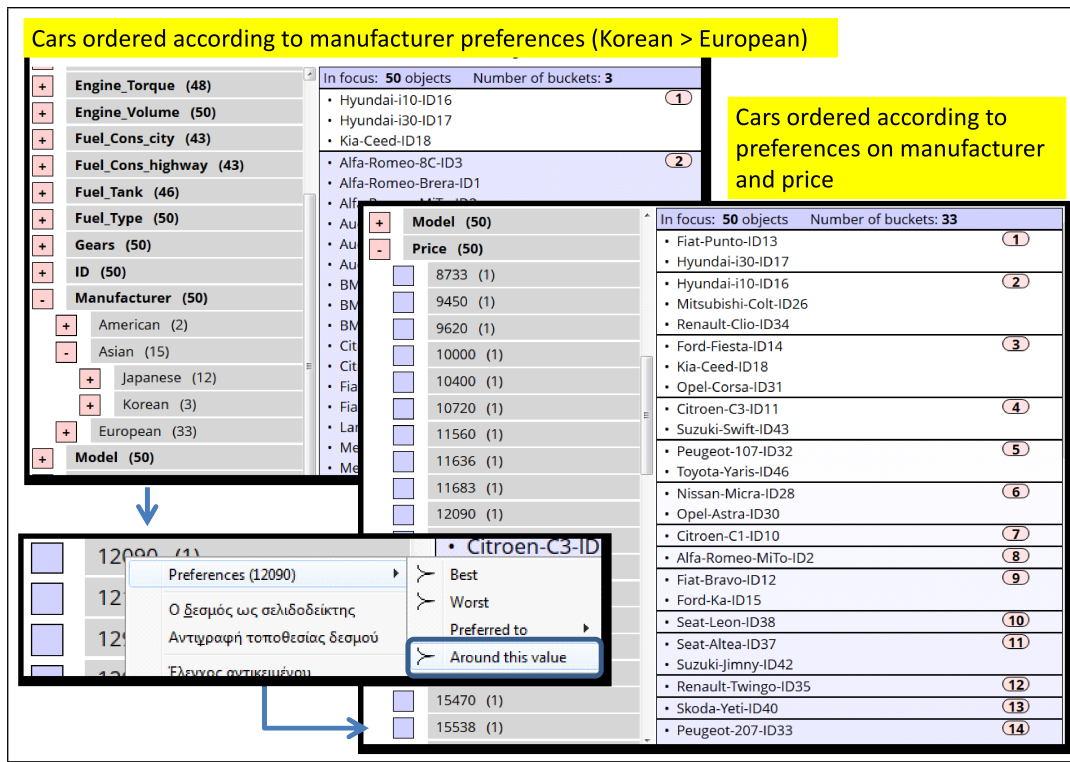


(b)

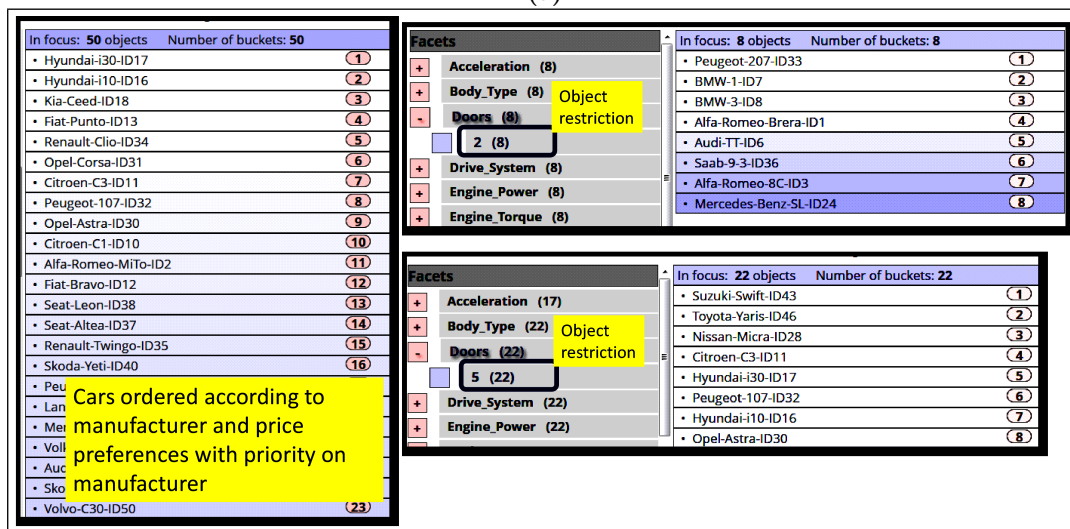


(c)

Figure 1: (a) Initial screen, (b) Value expansion - object restriction, (c) Relative preference  $Korean \succ European$



(a)



(b)

Figure 2: (a) Expressing preferences, (b) Object restrictions after preference expression

treatments and conditions, we used rotation and counterbalancing [6]. Specifically, we used a Graeco-Latin Square Design, rotating both the order of tasks and the order in which subjects experience the interfaces. For each task, an expert user provided the ordering of the collection according to preference by using the Hippalus system. The order was a bucket order (i.e. two cars can be equally preferred).

The users provided scores for the two UIs regarding *Ease of use*, *Usefulness*, *Preference* and *Satisfaction*, using a psychometric Likert scale from 1 to 5. We also calculated *Recall* (i.e. task completeness), *Precision*, and *Average Precision* of

the answer set, along with *Efficiency* (time to complete a task) and *Number of Actions* per each task using the logged data. Finally, users were asked explicitly if they prefer  $U_2$  over  $U_1$ . In case the answer was ‘Yes’, they were asked how much more useful they found  $U_2$  over  $U_1$  (*very much*, *much*, *enough*, or *little*).

**Results**

Here we synopsize the main results. All plain and expert users preferred the preferences UI over the plain one. Specifically, 75% of the 20 plain users found the  $UI_2$  to be *very*



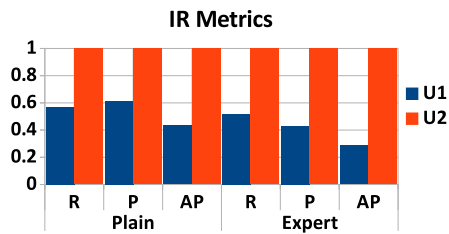


Figure 3: Average values in last step of each task for Recall (R), Precision (P) and Average Precision (AP)

much, 20% much and only 5% enough more useful than  $UI_1$ . The respective results for the 6 expert users are 50% very much and the other 50% much more useful. The preference-enabled UI, allowed users to complete all the tasks successfully, in average less than a third of the time and with a third of user interactions compared to the plain FDT UI (Fig. 3 and 4). Furthermore, none of the users was able to complete both of the tasks successfully with the plain UI. As a result we verify the conclusions of the theoretical user effort analysis in [15], since the preference-based UI helps users to find the desired results in less time and with fewer actions and less decisions. More details and a theoretical analysis of the user effort, decision cost and the gathered results are available in [9, Chapter 6].

## 5. CONCLUSION

In order to support decision making tasks, exploratory search requires a session-based behaviour that provides informative overviews. FDT is a widely used interaction model and in this paper we have presented an extension of this model with preferences. The enriched interaction is simple for the users, since it is mainly based on clicks over the presented values. In addition, the underlying preference framework is perfectly suited to FDT since it exploits the semantics of hierarchically organized values and automatically resolves any conflicts. This reduces the number of preference actions the users have to express and the dialogue is kept simple and clean (from technicalities). The Hippalus system demonstrates the feasibility of this extension. The results of the conducted user study were very satisfying: with the preference-enriched FDT all users completed all the tasks successfully in 1/3 of the time, performing 1/3 of the actions compared to the plain FDT.

## Acknowledgments

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operation Program "Education and LifeLong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Herakleitus II. Investing in knowledge society through the European Social Fund.

It was also partially supported by the PlanetData NoE (FP7:ICT-2009.3.4, #257641).

## 6. REFERENCES

- [1] D. Crawford, editor. *Supporting Exploratory Search*, volume 49. ACM, New York, NY, USA, 2006.

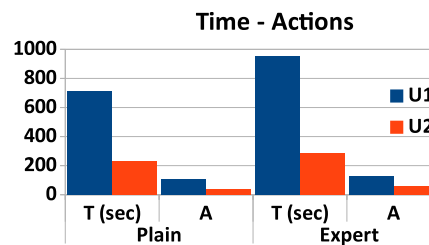


Figure 4: Average timings (T) and actions (A) per task

- [2] W. Dakka, P. Ipeirotis, and K. R. Wood. "Automatic Construction of Multifaceted Browsing Interfaces". In *Procs of CIKM'05*, pages 768–775, Nov. 2005.
- [3] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. "Dynamic Faceted Search for Discovery-Driven Analysis". In *Procs of CIKM'08*, 2008.
- [4] H. Inan. *Search Analytics: A Guide to Analyzing and Optimizing Website Search Engines*. Book Surge Publishing, 2006.
- [5] A. Kashyap, V. Hristidis, and M. Petropoulos. "FACeTOR: Cost-Driven Exploration of Faceted Query Results". In *Procs of CIKM'10*, pages 719–728. ACM, 2010.
- [6] D. Kelly. "Methods for Evaluating Interactive Information Retrieval Systems with Users". *Foundations and Trends in Information Retrieval*, 3(1-2):1–224, 2009.
- [7] I. Kitsos, K. Magoutis, and Y. Tzitzikas. Scalable entity-based summarization of web search results using mapreduce. *Distributed and Parallel Databases*, 2013.
- [8] J. Koren, Y. Zhang, and X. Liu. "Personalized Interactive Faceted Search". In *WWW'08: Procs of the 17th International Conference on World Wide Web*, pages 477–486, New York, NY, USA, 2008. ACM.
- [9] P. Papadakos. *Interactive Exploration of Multi-Dimensional Information Spaces with Preference Support*. PhD thesis, University of Crete, November 2013. Available at [http://www.ics.forth.gr/\\_publications/Papadakos\\_Dissertation.pdf](http://www.ics.forth.gr/_publications/Papadakos_Dissertation.pdf).
- [10] P. Papadakos, N. Armenatzoglou, S. Kopidaki, and Y. Tzitzikas. "On Exploiting Static and Dynamically Mined Metadata for Exploratory Web Searching". *Knowledge and Information Systems*, 30(3):493–525, 2012.
- [11] S. B. Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania. "Minimum-Effort Driven Dynamic Faceted Search in Structured Databases". In *Procs of CIKM'08*, pages 13–22, 2008.
- [12] G. M. Sacco and Y. Tzitzikas, editors. *Dynamic Taxonomies and Faceted Search: Theory, Practise and Experience*. Springer, 2009.
- [13] K. Stefanidis, G. Koutrika, and E. Pitoura. "A Survey on Representation, Composition and Application of Preferences in Database Systems". *ACM Transactions on Database Systems*, 36:19:1–19:45, August 2011.
- [14] M. Tvarožek, M. Barla, G. Frivolt, M. Tomša, and M. Bieliková. "Improving Semantic Search Via Integrated Personalized Faceted and Visual Graph Navigation.". In *SOFSEM*, volume 4910 of *Lecture Notes in Computer Science*, pages 778–789. Springer, 2008.
- [15] Y. Tzitzikas and P. Papadakos. Interactive exploration of multi-dimensional and hierarchical information spaces with real-time preference elicitation. *Fundamenta Informaticae*, 122(4):357–399, 2013.
- [16] A. Wagner, G. Ladwig, and T. Tran. "Browsing-Oriented Semantic Faceted Search". In *DEXA (1)*, pages 303–319, 2011.

# The DisC Diversity Model\*

Marina Drosou  
Computer Science & Engineering Dept.  
University of Ioannina, Greece  
mdrosou@cs.uoi.gr

Evaggelia Pitoura  
Computer Science & Engineering Dept.  
University of Ioannina, Greece  
pitoura@cs.uoi.gr

## ABSTRACT

In this paper, we summarize our work on diversification based on *dissimilarity* and *coverage* (*DisC* diversity) by presenting our main theoretical results and contributions.

## 1. DISC DIVERSITY

Diversification has attracted considerable attention, often as a means of enhancing the quality of the query results presented to users [3]. Most diversification approaches rely on assigning a diversity score to each data *item* and then selecting as diverse either the  $k$  items with the largest score for a given  $k$  (e.g., [1]), or the items with score larger than some predefined threshold (e.g., [9]).

In our work [4, 5], we address diversity through a different perspective and aim at selecting a representative subset that contains items that are *both* dissimilar with each other *and* cover the whole result set.

Let  $\mathcal{P}$  be a set of items. We define similarity between two items using a distance metric  $d$ . For a real number  $r$ ,  $r \geq 0$ , we use  $N_r(p_i)$  to denote the set of *neighbors* (or, the *neighborhood*) of an item  $p_i \in \mathcal{P}$ , i.e., the items lying at distance at most  $r$  from  $p_i$ :

$$N_r(p_i) = \{p_j \mid p_i \neq p_j \wedge d(p_i, p_j) \leq r\}$$

We use  $N_r^+(p_i)$  to denote the set  $N_r(p_i) \cup \{p_i\}$ . Items in the neighborhood of  $p_i$  are considered similar to  $p_i$ , while items outside its neighborhood are considered dissimilar to  $p_i$ . We define an  $r$ -DisC diverse subset as follows:

**DEFINITION 1. ( $r$ -DISC DIVERSE SUBSET)** *Let  $\mathcal{P}$  be a set of items and  $r$ ,  $r \geq 0$ , a real number. A subset  $S$  of  $\mathcal{P}$  is an  $r$ -Dissimilar-and-Covering diverse subset, or  $r$ -DisC diverse subset, of  $\mathcal{P}$ , if the following two conditions hold: (i) (coverage condition)  $\forall p_i \in \mathcal{P}$ ,  $\exists p_j \in N_r^+(p_i)$ , such that  $p_j \in S$  and (ii) (dissimilarity condition)  $\forall p_i, p_j \in S$  with  $p_i \neq p_j$ , it holds that  $d(p_i, p_j) > r$ .*

\*This work was supported by “Epirus on Android” a research project co-financed by the European Union (European Regional Development Fund-ERDF) and Greek national funds through the Operational Program “THESSALY-MAINLAND GREECE AND EPIRUS-2007-2013” of the National Strategic Reference Framework (NSRF 2007-2013)

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

The first condition ensures that all items in  $\mathcal{P}$  are represented by at least one similar item in  $S$  and the second condition that the items in  $S$  are dissimilar to each other. We call every item  $p_i \in S$  an  $r$ -DisC diverse item and  $r$  the *radius* of  $S$ . Instead of specifying a required size  $k$  of the diverse set or a threshold, our tuning parameter  $r$  explicitly expresses the degree of diversification and determines the size of the diverse set. Increasing  $r$  results in a smaller, more diverse subset, while decreasing  $r$  results in a larger, less diverse subset.

There may be more than one dissimilar and covering diverse subsets for the same set of items  $\mathcal{P}$ . Since we want a concise representation of  $\mathcal{P}$ , we select the smallest one:

**DEFINITION 2. (MINIMUM  $r$ -DISC DIVERSE SUBSET PROBLEM)** *Given a set  $\mathcal{P}$  of items and a radius  $r$ ,  $r \geq 0$ , find an  $r$ -DisC diverse subset  $S^*$  of  $\mathcal{P}$ , such that, for every  $r$ -DisC diverse subset  $S$  of  $\mathcal{P}$ , it holds that  $|S^*| \leq |S|$ .*

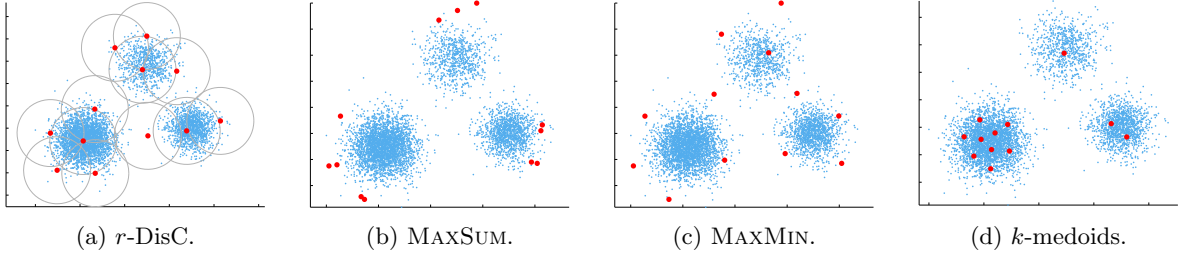
It has been shown that any  $r$ -DisC diverse subset  $S$  of  $\mathcal{P}$  is at most  $B$  times larger than any minimum  $r$ -DisC diverse subset  $S^*$ , where  $B$  is the maximum number of independent (i.e., dissimilar to each other) neighbors of any item in  $\mathcal{P}$  [4].  $B$  depends on the distance metric used and on the dimensionality of the data space. In many cases,  $B$  is a constant, e.g., for the 2D Euclidean plane,  $B = 5$ .

**Comparison with Other Models.** Let us now compare DisC with two widely used diversification models, namely MAXMIN and MAXSUM, that aim at selecting a subset  $S$  of  $\mathcal{P}$  so as the minimum or the average pairwise distance of the selected items is maximized (e.g., [7, 8, 2]). We also compare DisC with  $k$ -medoids, a widespread clustering algorithm. In this case, the located medoids constitute the representative subset  $S$ . Input in all the above approaches is the size  $k$  of the diverse subset  $S$ . Figure 1 shows the corresponding sets attained by first locating an  $r$ -DisC diverse subset for a given  $r$  and then using the size of the produced diverse subset as the input  $k$  of the other approaches. Here,  $r = 0.15$  and  $k = 12$ .

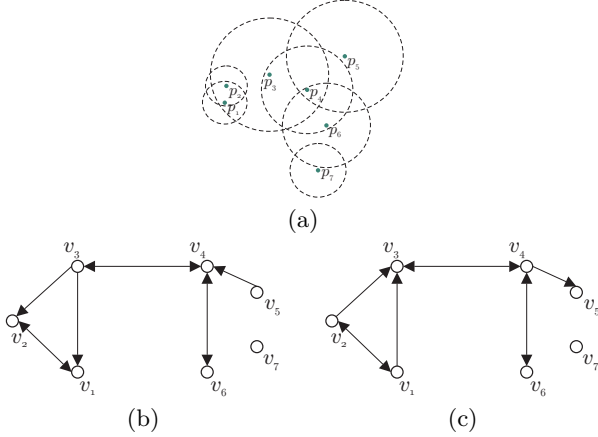
MAXSUM and  $k$ -medoids fail to cover all areas of the dataset; MAXSUM focuses on the outskirts of the dataset, whereas  $k$ -medoids reports only central items, ignoring items that are further away. MAXMIN performs better in this aspect. However, since MAXMIN seeks to retrieve items that are as far apart as possible, it fails to retrieve items from dense areas. DisC avoids most of these problems.

**Multiple Radii.** There may be cases in which we want different parts of the data space to be represented with more





**Figure 1: Diverse subsets of size  $k = 12$  produced by different diversification methods for a clustered dataset. Selected items are shown as (red) solid circles. Circles around items of the DisC solution denote the radius  $r$  of the selected items.**



**Figure 2: (a) A set of items associated with different radii and their graph representation for the (b) Covering and (c) CoveredBy problems. A directed edge from  $v_i$  to  $v_j$  indicates that  $d(p_i, p_j) \leq r(p_i)$  and  $d(p_i, p_j) \leq r(p_j)$  respectively.**

or less items. Thus, we consider the more general case where each item  $p_i$  is associated with a different radius  $r(p_i)$ .

The problem now loses its symmetry, since an item  $p_i$  may be in the neighborhood of an item  $p_j$ , while  $p_j$  is not in the neighborhood of  $p_i$ . This gives rise to two different interpretations of radius. One interpretation is that  $p_i$  can represent all items in its neighborhood. The other interpretation is that  $p_i$  can be represented by all items its neighborhood. We call the first problem *Covering DisC diverse subset problem* and the second one *CoveredBy DisC diverse subset problem*.

**DEFINITION 3. (COVERING (RESP. COVEREDBY) DISC DIVERSE SUBSET)** Let  $\mathcal{P}$  be a set of items and  $r : \mathcal{P} \rightarrow \mathbb{R}^+$  be a function determining the radius of each item in  $\mathcal{P}$ . A subset  $S$  of  $\mathcal{P}$  is a *Covering (resp. CoveredBy) Dissimilar-and-Covering diverse subset*, or *Covering (resp. CoveredBy) DisC diverse subset*, of  $\mathcal{P}$ , if the following two conditions hold: (i) (coverage condition)  $\forall p_i \in \mathcal{P}, \exists p_j \in S$  with  $d(p_i, p_j) \leq r(p_j)$  (resp.  $d(p_i, p_j) \leq r(p_i)$ ), such that  $p_j \in S$  and (ii) (dissimilarity condition)  $\forall p_i, p_j \in S$  with  $p_i \neq p_j$ , it holds that  $d(p_i, p_j) > \max\{r(p_i), r(p_j)\}$ .

Figure 3 presents a qualitative view of various options of assigning radii to items. We present three different scenarios. The first one corresponds to the case where some parts of the dataset are considered more important than others and we want them to be represented with more items. In Figure 3a,

items in each of the four quadrants are assigned increasing radii as we move clockwise. The second scenario corresponds to the case in which we want to take into account density, so that dense areas are not under-represented in the diverse subset. In this case, we assign smaller radii to items in denser areas (Figure 3b). The third scenario corresponds to the case in which we want to relate representation with relevance. For example, for the CoveredBy problem, we assign smaller radii to items with larger relevance (Figure 3c and Figure 3d). This ensures that each item can be covered only by items that have a larger relevance than it.

**Graph Representation and NP-hardness.** Besides the geographical interpretation of DisC diversity, there is also a corresponding graph representation. We define next the corresponding graph models for both the single and the multiple radii cases.

For a single radius  $r$ , let  $G_{\mathcal{P}, r} = (V, E)$  be an undirected graph such that there is a vertex  $v_i \in V$  for each item  $p_i \in \mathcal{P}$  and an edge  $(v_i, v_j) \in E$ , if and only if,  $d(p_i, p_j) \leq r$  for the corresponding items  $p_i, p_j$ . Considering multiple radii, let  $G_{\mathcal{P}, r(\cdot)} = (V, E)$  be a directed graph such that there is a vertex  $v_i \in V$  for each item  $p_i \in \mathcal{P}$  and a (directed) edge  $(v_i, v_j) \in E$ , if and only if, for the corresponding items  $p_i, p_j$ , it holds that  $d(p_i, p_j) \leq r(p_i)$  (Covering problem) or  $d(p_i, p_j) \leq r(p_j)$  (CoveredBy problem). An example is shown in Figure 2.

It turns out that DisC diverse subsets correspond to independent and dominating sets of the corresponding graphs. A *dominating* set  $D$  for a graph  $G$  is a subset of vertices of  $G$  such that every vertex of  $G$  not in  $D$  is joined to at least one vertex in  $D$  by some edge when  $G$  is undirected and by an incoming edge when  $G$  is directed. An *independent* set  $I$  for a graph  $G$  is a set of vertices of  $G$  such that for every two vertices in  $I$ , there is no edge connecting them. Intuitively, a dominating set of  $G_{\mathcal{P}, r}$  satisfies the covering condition of the DisC diverse subset, whereas an independent set of  $G_{\mathcal{P}, r}$  satisfies the dissimilarity condition of the DisC diverse subset.

**LEMMA 1.** Finding a DisC diverse subset for a set  $\mathcal{P}$  is equivalent to finding an independent dominating set of the corresponding graph  $G$ .

Finding a minimum independent dominating set of a graph has been proven to be NP-hard (e.g., [6]).

**Computing DisC Diverse Subsets.** Next, we present a general algorithm for locating DisC diverse subsets (Algorithm 1). For presentation convenience, let us call *black* the items of  $\mathcal{P}$  that are in the diverse subset  $S$ , *grey* the

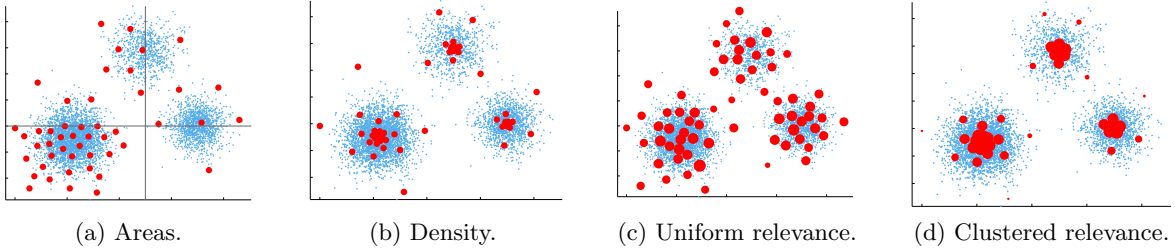


Figure 3: Using multiple radii. Selected items are shown as (red) solid circles.

---

**Algorithm 1** Locating DisC diverse subsets.

---

**Input:** A set of items  $\mathcal{P}$ , a radius function  $r(\cdot)$  and a selection criterion  $\mathcal{C}(\cdot)$ .

**Output:** A DisC diverse subset  $S$  of  $\mathcal{P}$ .

---

```

1:  $S \leftarrow \emptyset$ 
2: for all  $p_i \in \mathcal{P}$  do
3:   color  $p_i$  white
4: end for
5: while there exist white items do
6:   select the white item  $p_i$  with the largest value of  $\mathcal{C}(p_i)$ 
7:    $S = S \cup \{p_i\}$ 
8:   color  $p_i$  black
9:   for all  $p_j \in N_{r(p_i)}^W(p_i)$  (Covering) or  $p_j$  s.t.  $p_i \in N_{r(p_j)}(p_j)$  (CoveredBy) do
10:    color  $p_j$  grey
11:   end for
12: end while
13: return  $S$ 

```

---

items covered by some item in  $S$  and *white* the items that are neither black nor grey.  $N_r^W(p_i)$  denotes the set of white neighbors of  $p_i$ . Initially,  $S$  is empty and all items are white. Items are selected for inclusion in  $S$  in rounds based on some selection criterion  $\mathcal{C}$ .

For the single radius case, selecting at each round any white item will result in a DisC diverse subset. In addition, the greedy algorithm that selects at each round the white item  $p_i$  with the largest white neighborhood  $N_r^W(p_i)$  results in DisC diverse subsets with size close to the minimum one [4]. For the multiple radii case, to attain DisC diverse items, we need to select white items in decreasing order of their radius for the Covering problem and in increasing order of their radius for the CoveredBy problem.

**Zooming.** We also consider a *zooming* operation where, after being presented with an initial set of results for some radius  $r$ , a user asks to see either more or less results by correspondingly decreasing or increasing the radius. For simplicity, we shall focus on zooming in the case of a single radius. Formally, given a set of items  $\mathcal{P}$  and an  $r$ -DisC diverse subset  $S$  of  $\mathcal{P}$  for some specific radius, we want to compute an  $r'$ -DisC diverse subset  $S'$  of  $\mathcal{P}$ . There are two cases: (i)  $r' < r$  (*zooming-in*) and (ii)  $r' > r$  (*zooming-out*). Ideally,  $S' \supseteq S$ , for  $r' < r$  and  $S' \subseteq S$ , for  $r' > r$ .

To study the relationship between  $S$  and  $S'$ , for two radii  $r_1, r_2, r_2 \geq r_1$ , we define the set  $N_{r_1, r_2}^I(p_i)$ , as the set of items at distance at most  $r_2$  from  $p_i$  which are at distance at least  $r_1$  from each other.  $|N_{r_1, r_2}^I(p_i)|$  can be bounded for specific distance metrics and dimensionality [4].

When zooming-in, we construct diverse sets that are supersets of  $S$  by adding items to  $S$ . It holds that:

LEMMA 2. For zooming-in: (i)  $S \subseteq S'$  and (ii)  $|S'| \leq |S| + \sum_{p_i \in S} |N_{r', r}^I(p_i)|$

When zooming-out, it may not be possible to construct a DisC diverse subset  $S'$  that is a subset of  $S$ . Thus, we proceed in two passes. In the first pass, we examine all items of  $S$  in some order and remove their diverse neighbors that are now covered by them. At the second pass, items from any uncovered areas are added to  $S'$ . It holds that:

LEMMA 3. For zooming-out: (i) There are at most  $\sum_{p_i \in S} |N_{r, r'}^I(p_i)|$  items in  $S \setminus S'$ , (ii) For each item of  $S$  not included in  $S'$ , at most  $B - 1$  items are added to  $S'$ .

## 2. SUMMARY AND FUTURE WORK

In a nutshell, we introduced a new, intuitive definition of diversity based on using a radius  $r$  rather than a size limit  $k$ . We presented both a geometrical and an equivalent graph-based interpretation of our model. We introduced incremental diversification through zooming-in and zooming-out, showed that locating DisC diverse subsets is an NP-hard problem and provided efficient algorithms for their computation. Directions for future work include extending our approach to the budgeted  $r$ -DisC problem, that is, computing DisC subsets of a specific size that maximize coverage and also studying different variations of our zooming operations.

## 3. REFERENCES

- [1] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD Conference*, 2011.
- [2] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, pages 155–166, 2012.
- [3] M. Drosou and E. Pitoura. Search result diversification. *SIGMOD Record*, 39(1):41–47, 2010.
- [4] M. Drosou and E. Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *PVLDB*, 6(1):13–24, 2012.
- [5] M. Drosou and E. Pitoura. Poikilo: A tool for evaluating the results of diversification models and algorithms. *PVLDB*, 6(12):1246–1249, 2013.
- [6] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [7] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.
- [8] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. On query result diversification. In *ICDE*, pages 1163–1174, 2011.
- [9] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, pages 368–378, 2009.

# Exploring RDF/S Evolution using Provenance Queries

Haridimos Kondylakis

Dimitris Plexousakis

Institute of Computer Science, FORTH  
Vasilika Vouton, Heraklion  
Greece

kondylak@ics.forth.gr

dp@ics.forth.gr

## ABSTRACT

The evolution of ontologies is an undisputed necessity in current research community. The problem of understanding this evolution is a fundamental problem as, based on this understanding, maintainers of depending artifacts need to take a decision about possible changes. Moreover, as ontologies are often developed by several ontology engineers, it is also important for them to understand what changes have been made by each other. Recent research focuses on just identifying and presenting the changes from one ontology version to another. In this paper, we argue that this is not enough and that we need more fine-grained methods for understanding how the ontology evolved. To this direction, we present a module, named *ProvenanceTracker*, which gets as input the list of changes between two or more RDF/S ontology versions and can answer fine-grained provenance queries about ontology resources. Our module can identify *when* a resource was created and *how*. The sequence of changes that led to the creation of that specific resource can be identified and presented to the user. We evaluate the time complexity of our approach and show that it can possibly reduce the human effort spent on understanding ontology evolution.

## Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

## General Terms

Algorithms, Experimentation, Languages, Theory.

## Keywords

Ontology Evolution, Provenance

## 1. INTRODUCTION

Ontologies are defined as formal, explicit specification of a shared conceptualization of a domain of interest [1]. However ontologies are not static but they are living artifacts and subject to change [2]. Due to the rapid development of research, ontologies are frequently changed to depict the new knowledge that is acquired. Since ontologies are usually managed independently from each other they can be used and extended without the explicit permission of the owner. In several cases, the owner of an ontology is completely unaware of who uses or extends his

ontology.

It is therefore vital to be able to support the ontology engineers and the maintainers of the dependent artifacts in this complex process of ontology evolution [3]. Several approaches so far deal with problems such as consistency maintenance, backward compatibility, ontology manipulation, change propagation, etc. [2]. In the field of *a posteriori* understanding ontology evolution most of the approaches use different representation languages to model ontology evolution [4] that they just present to the users. However, although the languages of changes used have become more concise and compact - by employing high-level change operators (operators that can describe complex updates, e.g. the insertion of an entire sub-sumption hierarchy) - still ontology understanding relies on just presenting to the users a huge list of changes between ontology versions.

In this paper, we argue that only listing the changes between two versions is insufficient for the purpose of understanding ontology evolution. Moreover, we provide a solution to this problem by answering provenance queries concerning both the data and the schema information of an ontology. To that direction, we present a module, named *ProvenanceTracker* that gets as input two or more ontology versions and it is able to answer queries requesting fine-grained provenance information. In order to do that, a preprocessing step is required that automatically generates the “on-the-fly” the sequence of changes between those versions. This is accomplished by employing an external module, described extensively in [4], which gets as input subsequent ontology versions and produces automatically the sequence of changes between them.

In our approach we define the notions of *how* and *when provenance* and we present the corresponding algorithms. Using our module a user can identify with which change operation a resource was introduced (*how*) and in which ontology version (*when*). Moreover, the list of change operations that led to the creation of that specific resource can be computed and presented to the user (*extended-how*) allowing further exploration. This knowledge can be used to drive developer’s understanding on ontology evolution for that specific resource. The simplicity of our approach makes it a valuable tool for ontology engineers and provides a unique vantage point on long and complex evolution histories.

Finally, we describe our implementation and we present our experimental analysis using two well-known ontologies CIDOC-CRM [5] and Gene Ontology [6]. Experiments performed show the feasibility of our approach and the considerable advantages gained.

The rest of the paper is organized as follows: Section 2 presents related work and Section 3 provides preliminaries and introduces the problem by an example. Then, Section 4 presents the

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

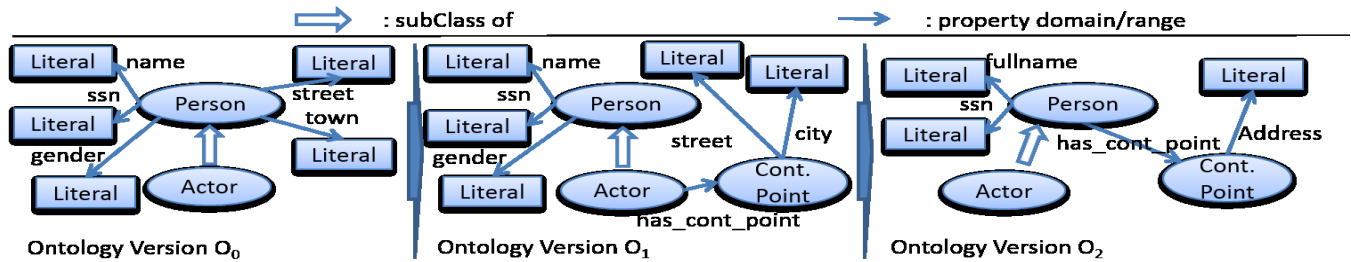


Figure 1. Example Ontology Evolution

algorithms for answering provenance queries about ontology evolution. Section 5 describes the implemented system and our experimental analysis. Finally, Section 6 provides a summary and an outlook for further research.

## 2. RELATED WORK

Management of provenance information has been extensively studied in the literature, using different methods and approaches. Different authors define different provenance management techniques (e.g. *Why-provenance* [7], *Trio-Provenance* [8], provenance *semi-rings* [9]) that either try to provide annotations at the tuple level or to extract provenance information by analysing queries. Other works such as [10] try to describe the relationship between source and target data in a data integration scenario. However our approach differs in both methods and goals. To the best of our knowledge, there is no other approach that tries to answer provenance queries on ontology evolution.

Other works that could be employed to understand ontology evolution focus on change detection. Those systems can be classified under two basic dimensions, namely the level of changes they support (low-level or high-level) and the underlying representation language assumed (Description Logic [3], RDF/S [4] etc.). In its simplest form, a language of changes consists of only two *low-level* operations, *Add(x)* and *Delete(x)*, which determine individual constructs (e.g., triples) that were added or deleted [11, 12]. However, a significant number of recent works [4, 12-15] imply that *high-level* change operations should be employed instead, which describe more complex updates, as for instance the insertion of an entire subsumption hierarchy. A high-level language is preferable than a low-level one [16], as it is more *intuitive*, *concise*, *closer to the intentions* of the ontology editors and *captures more accurately* the semantics of change. However, there is no agreed-upon list of changes that are necessary for any given context. In our case, we do not redefine such a language but we only use one of them. Moreover, our results are not limited to this specific language as we shall see later in this paper.

A similar approach to ours, is in [3] where a change is defined and detected using temporal queries over a version log that contains recordings of the applied changes. However, the version log must be updated whenever a change occurs. This overrules the use of this approach in non-curated or distributed environments. In our approach, on the other hand, the changes can be produced a posteriori and no temporal queries are used.

In [17] the authors provide a mechanism to document schema evolution of relational DBs, by presenting automatically the changes (called SMOs) between those versions. However, those changes are not detected fully automatically. Moreover, they offer

a schema evolution history analysis tool, but this tool only provides coarse-grained results.

Finally, in [18] the authors present a tool to allow several developers to make changes concurrently and remotely to the same ontology, track changes, and manage ontology versions. However, this tool focuses on conflict resolution and cannot provide answers to fine-grained provenance queries.

## 3. PRELIMINARIES & MOTIVATING EXAMPLE

RDF is a language for describing web resources [19]. Information in RDF is represented using triples of the form (*subject*, *predicate*, *object*) which record that *subject* is related to *object* via *predicate*. RDF datasets have attached semantics through RDFS schemas [19]. RDFS is a vocabulary description language that includes a set of inference rules use to generate new, implicit triples from explicit ones. Most of the Semantic Web Schemas (85,45%) are expressed in RDF/S [20] and RDF/S offers, in our case, an optimum trade-off between expressive power and efficient reasoning support. In this paper we restrict ourselves to *valid RDF/S knowledge bases*. The validity constraints [4] that we consider in this work concern mostly the *type uniqueness*, i.e., that each resource has a unique type, the *acyclicity* of the *subClassOf* and *subPropertyOf* relations and that the subject and object of the instance of some property should be correctly classified under the domain and range of the property, respectively. Those constraints are enforced in order to enable *unique* and *non-ambiguous* detection of the changes among the ontology versions as we shall later discuss.

Now as an example, consider an ontology shown on the left of Figure 1 (ontology version  $O_0$ ). This ontology is used as a point of common reference, describing people and their contact points. Assume now that at some point in time, the ontology evolves and we get  $O_1$  by adding the class “*Cont.Point*” describing contact points and the property “*has\_cont\_point*” between the class “*Actor*” and the class “*Cont.Point*”. Moreover, the domain of the literals “*street*” and “*city*” is changed to the class “*Cont.Point*”. Then the ontology designer decides to evolve again the ontology and to produce  $O_2$ . So, the domain of the “*has\_cont\_point*” property is moved from the class “*Actor*” to the class “*Person*”, and the property “*gender*” is deleted. Moreover, the “*street*” and the “*city*” properties are merged to the “*address*” property as shown on the right of Figure 1. For modeling this evolution we use the language of changes and the corresponding detection algorithm as proposed in [4]. The language contains over 70 types of change operations and three of them are described in Figure 2.

Change	Generalize Domain(a,b,c)	Rename Property(a,b)	Merge Properties(A,b)
Intuition	Change the domain of prop. $a$ from $b$ to superclass $c$	Rename property $a$ to $b$	Merge properties contained in $A$ into $b$
$\delta_a$	$[(a, \text{domain}, c)]$	$[(b, \text{type}, \text{property})]$	$(b, \text{type}, \text{property})$
$\delta_d$	$[(a, \text{domain}, b)]$	$[(a, \text{type}, \text{property})]$	$\forall a \in A : (a, \text{type}, \text{property})$
Inverse	Specialize Domain(a,c,b)	Rename Property(b,a)	Split Property(b, A)

Figure 2. Example change operations

A change operation is defined as follows:

**Definition 3.1** (Change Operation): *A change operation  $u$  over an RDF ontology  $O$ , is any tuple  $(\delta_a, \delta_d)$  where  $\delta_a \cap O = \emptyset$  and  $\delta_d \subseteq O$ . A change operation  $u$  from  $O_1$  to  $O_2$  is a change operation over  $O_1$  such that  $\delta_a \subseteq O_2 \setminus O_1$  and  $\delta_d \subseteq O_1 \setminus O_2$ .*

Obviously,  $\delta_a$  and  $\delta_d$  are sets of triples. For simplicity we will denote  $\delta_a(u)$  the added and  $\delta_d(u)$  the deleted triples of a change  $u$ . From the definition, it follows that  $\delta_a(u) \cap \delta_d(u) = \emptyset$  and  $\delta_a(u) \cup \delta_d(u) \neq \emptyset$  if  $O_1 \neq O_2$ . The application of a change  $u$  over an ontology version  $O$ , denoted by  $u(O)$ , is defined as  $u(O) = (O \cup \delta_a(u)) \setminus \delta_d(u)$ . Moreover the application of a sequence of change operations  $us$  to an ontology, i.e.  $us(O)$ , is defined as the sequential application of the change operation in  $us$  to  $O$ . An important note for those change operations is that for any two changes  $u_1, u_2$  in such a sequence it holds that  $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$  and  $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$ . As a consequence the sequence of changes between two ontology versions is unique. The interested reader is forwarded to [4] for more information on the aforementioned language of changes.

In our example the change log between  $O_0$  and  $O_1$ , i.e. the  $E^{O_0, O_1}$ , consists of the following change operations:

- $u1: \text{Add\_Class}(\text{Cont.Point}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$
- $u2: \text{Add\_Property}(\text{has\_cont\_point}, \emptyset, \emptyset, \emptyset, \emptyset, \text{Actor}, \text{Cont.Point}, \emptyset, \emptyset)$
- $u3: \text{Move\_Property}(\text{town}, \text{Person}, \text{Cont.Point})$
- $u4: \text{Move\_Property}(\text{street}, \text{Person}, \text{Cont.Point})$
- $u5: \text{Rename\_Property}(\text{town}, \text{city})$

Moreover, the change log  $E^{O_1, O_2}$  consists of the following change operations:

- $u6: \text{Delete\_Property}(\text{gender}, \emptyset, \emptyset, \emptyset, \emptyset, \text{Person}, \text{xsd:String}, \emptyset, \emptyset)$
- $u7: \text{Generalize\_Domain}(\text{has\_cont\_point}, \text{Actor}, \text{Person})$
- $u8: \text{Merge\_Properties}(\{\text{street}, \text{city}\}, \text{address})$
- $u9: \text{Rename\_Property}(\text{name}, \text{fullname})$

Obviously,  $E^{O_0, O_2} = E^{O_0, O_1} \cup E^{O_1, O_2}$ . In this paper we argue that only presenting the above sequence of changes is not enough for understanding how ontology evolved. Especially in real world scenarios, the large number of change operations makes it impossible for ontology developers to understand ontology evolution based solely on those. In our experiments for example, we had 4175 changes for Gene Ontology and 726 changes for CIDOC-CRM.

To overcome this problem we designed and implemented the *ProvenanceTracker* module. This module augments the understanding of ontology evolution by answering a range of provenance queries, including the following ones: How was a resource added to the ontology? By which change operation was the “Address” literal added? What are the change operations that had some influence on the creation of the “Address” literal? When was the “Address” literal added to the ontology?

Similar terminology [21] is widely used in relational environments. However, to the best of our knowledge it is the first time that we use this terminology to capture provenance

information on ontology evolution. Moreover, although our ontology and change operations can be used on instance level as well, in this paper we will focus only on schema level without loss of generality.

## 4. QUERIES ON SCHEMA PROVENANCE

As already mentioned, presenting only the list of changes between ontology versions is not adequate for understanding ontology evolution. To answer queries about how a specific resource was added we define the notion of an affecting change operation.

**Definition 4.1** (Affecting Change Operation). *Let  $r$  be a resource of an ontology version  $O_m$  and  $E^{O_k, O_m}$  ( $k < m$ ) the sequence of changes between  $O_k$  and  $O_m$ . A change operation  $u \in E^{O_k, O_m}$  affects the resource  $r$ , denoted by  $\text{aff}(r)$ , if  $r \in O_m$  and  $r \in \delta_a(u)$ .*

An affecting change operation captures the way a resource was introduced in the ontology. Assuming that we have  $E^{O_k, O_m}$  already constructed it is quite easy to identify the affecting change operation by just scanning the change log once. We have to note that the affecting change operation if it exists is *unique*. This is due to the fact that for our languages of changes it holds the following: for any two changes  $u_1, u_2$ ,  $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$  and  $\delta_d(u_1) \cap \delta_d(u_2) = \emptyset$  as described in Section 2. In our example the query *how*(“Address”) when applied in  $E^{O_0, O_2}$  it will return the change operation  $u8: \text{Merge\_Properties}(\{\text{street}, \text{city}\}, \text{address})$ . In the case that no affecting change operation is found, no answer will be returned. This means that the aforementioned resource was added before  $O_k$  and we have no information about it.

Now we would like to know in which ontology version the “Address” resource was introduced. The idea is similar to answering *how* provenance queries. We only have to scan once the change log  $E^{O_k, O_m} = E^{O_k, O_{k+1}} \cup \dots \cup E^{O_{m-1}, O_m}$ . If  $\text{aff}(r) \in E^{O_k, O_m}$  then obviously  $r \in O_m$  so we can conclude that  $r$  was introduced in  $O_m$ . In our example the query *when*(“Address”) will return  $O_2$  as an answer.

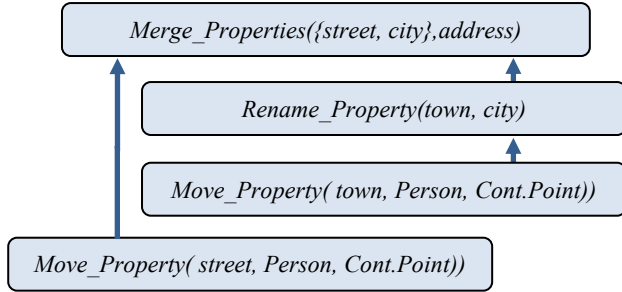
Presenting only the affecting change operations and the ontology version that a resource has been introduced does not necessarily provide insights on the corresponding ontology evolution. When drastic evolution occurs, those are not enough and we would like to get more information about which part of the ontology evolved to produce the specific resource. So, instead of providing just the affecting change operation and the ontology version, our idea is to present the history of the evolution of the specific parts of the ontology as an answer to *extended-how* provenance queries.

For example, by checking the change log  $E^{O_0, O_2}$  presented on Section 3, we can easily identify that the operations shown in Figure 3, describe the evolution of the “address” resource.

Presenting such a graph to the ontology engineers, their understanding on the ontology evolution is focused on the specific parts that evolved to produce the aforementioned resource. Such a sequence of change operations that depict the history of the ontology with respect to a specific resource  $r$  is called a *change path* for that resource. However, before defining the change path



for a given resource we will define the change path for a *change operation* first.



**Figure 3. The change path for the “Address” resource visualized as a tree.**

**Definition 4.2** (Change path for a change operation). A change path for the change operation  $u \in E^{Ok,Om}$ , ( $k < m$ ) denoted by  $us_{path}^u$ , is the minimal sequence of change operations in  $E^{Ok,Om}$  such that  $u \in us_{path}^u$  and that  $us_{path}^u(O_k) \subseteq O_m$ .

A change path is *minimal* in the sense that one cannot remove any of the change operations in it and still  $us_{path}^u(O_k) \subseteq O_m$ . The change path presents the history of the evolution of the specific part of the ontology for a specific change operation. For example, the change path for the change  $u_8$ :  $Merge\_Properties(\{street, city\}, address)$  is  $us_{path}^{u_8} = [u_3, u_4, u_5, u_8]$  as shown in Figure 3 and  $us_{path}^{u_8}(O_0) \subseteq O_2$ .

**Proposition 2** (Uniqueness): The change path  $us_{path}^u$  over  $E^{Ok,Om}$  is unique.

*Proof:* Assume  $us_{path}^u$  is not unique. This would mean that we can have two change paths  $us_{path1}^u$  and  $us_{path2}^u$ . Since they are both change paths it should hold that  $size(us_{path1}^u) = size(us_{path2}^u)$  since they both have to be minimal. Now let  $us_{path1}^u = [u_{k1}, \dots, u_{kn}]$  and  $us_{path2}^u = [u_{m1}, \dots, u_{mn}]$ . Since they are both change paths  $u = u_{kn} = u_{mn}$ . For  $i < n$ , each one of the  $u_{ki}, u_{mi}$  deletes a part of the ontology and adds another part. Since the two change paths have the same minimal size and  $u = u_{kn} = u_{mn}$  in order to be different there must exist two change operations  $u_{ki}, u_{mj}$  such that  $u_{ki} \neq u_{mj}$  and  $\delta_a(u_{ki}) \cap \delta_a(u_{mj}) \neq \emptyset$  since they should delete a common part of the ontology. However, this is impossible since  $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$  for our change operations.

**Algorithm 4.1:**  $ComputeChangePath(E^{Ok,Om}, u)$   
**Input:** A sequence  $E^{Ok,Om} = [u_1, \dots, u_n]$  and one change operation  $u$   
**Output:** a sequence of change operations  $us'$   
1.  $us' := u$   
2. For  $i=n$  to 1  
3. if there exists  $t \in \delta_a(u_i)$  such that  $t \in \delta_a(us')$   
4.  $us' := us' \cup u_i$   
5. Return  $us'$

**Figure 4. Computing the change path for a given change operation.**

Now, we will present an algorithm that given a change log produces the change path for a change operation  $u$ . The algorithm is shown in Figure 4. The idea is the following: The algorithm starts from the input change operation and identifies the triples that are added to the ontology, possibly by deleting other triples. Then it searches for the change operations that delete that added

information in order to add new information and so on. After the execution the change path for  $u$  will be stored in  $us'$ .

**Theorem 1:** The algorithm  $ComputeChangePath$  computes  $us_{path}^u$  over  $E^{Ok,Om}$ .

*Proof:* In order to prove that algorithm  $ComputeChangePath$  computes the change path for a given change operation  $u$  over a change log  $E^{Ok,Om}$  we have to prove that (a)  $u \in us'$ , (b)  $us'(O_k) \subseteq O_m$  and that (c)  $us'$  is minimal.

(a) From line 1 of the algorithm indeed  $u \in us'$ .

(b) Let's assume that  $us'(O_k)$  is not a subset of  $O_m$ . This would mean that there exists at least one triple, assume  $t'$  in  $us'(O_k)$  such that it does not exist in  $O_m$ . So, to reach  $O_k$ , there should be a change operation  $u'$  such that  $t' \in \delta_a(u')$  such that  $t' \in \delta_a(us')$  not identified by our algorithm. However this is impossible from line 3 of our algorithm.

(c) Now we prove minimality. Let's assume that  $us'$  is not minimal. This would mean that there is  $us_{path}$  with  $size(us_{path}) < size(us')$ . This would mean that there exist  $u' \in us'$  such that  $u' \notin us_{path}$ . Of course this would mean from lines 3 and 4 that there exist  $t'$  such that  $t' \in \delta_a(u')$  and  $t' \in \delta_a(us')$ . However, this would mean that  $t'$  does not belong to  $O_m$ , and should be deleted by another change operation. However for our change operations it holds that  $\delta_a(u_1) \cap \delta_a(u_2) = \emptyset$  which makes the previous statement impossible. So  $us'$  is minimal as well.

The time complexity of the algorithm is  $O(N * M * S)$  where  $N$  is the number of change operations,  $M$  the maximum size of triples in a change operation  $u$  and  $S$  the number of triples in  $\delta_a(us')$ . Moreover, it is easy to change Algorithm 4.1 in order to retrieve the change path for a given resource. This will allow the developers to examine the evolution of the ontology concerning a specific resource:

**Definition 4.3** (Change path for a resource). The change path  $us_{path}$  over  $E^{Ok,Om}$  for the resource  $r \in O_m$  is  $us_{path}^r = \cup us_{path}^{u_i}$ ,  $r \in u_i$ .

The algorithm is shown in Figure 5. The idea is that we would like to retrieve the history of the evolution of resource  $r$ . However,  $r$  might appear in several triples so we need to identify all change paths that have to do with it.

**Theorem 2:** The algorithm  $ComputeChangePathTriple$  computes the change path for a given resource  $r$  over  $E^{Ok,Om}$ .

**Algorithm 4.2:**  $ComputeChangePathResource(E^{O1,O2}, r)$   
**Input:** A sequence  $E^{O1,O2} = [u_1, \dots, u_n]$  and one resource  $r$   
**Output:** a sequence of change operations  $us'$   
1.  $us' := \emptyset$   
2. For  $i=n$  to 1  
3. If  $t \in \delta_a(u_i)$  such that  $r \in t$   
4.  $us' := us' \cup ComputeChangePath(E^{O1,O2}, u_i)$   
5. Return  $us'$

**Figure 5. Computing the change path for a given resource.**

The algorithm is immediately proved by construction. Algorithm 4.2 needs to scan the change log one time per triple containing the resource  $r$  in order to identify the change operation that inserts the given resource. So the complexity of the algorithm becomes  $O(T * N * M * S)$  where  $T$  is the number of triples containing  $r$ ,  $N$  is the number of change operations,  $M$  the maximum size of triples in a change operation  $u$  and  $S$  the number of triples in  $\delta_a(us')$ .

## 5. IMPLEMENTATION & EVALUATION

The *ProvenanceTracker* module described in this paper is implemented as a module of our *Exelixis* platform (<http://139.91.183.29:8080/exelixis/>). The platform uses JAVA for the algorithms and HTML/JQuery for the presentation layer. Using the *Exelixis* platform, a user is able to load an RDF/S ontology, to visualize and explore it. Furthermore, as more ontology versions become available, the change logs between them are automatically constructed and stored to the system. Then, a user can issue queries - denoting the ontology version that those queries are using- which are being forwarded to the underlying data integration engines to be answered. The system automatically identifies registered data integration systems that might use different ontology versions and tries to produce equivalent query rewritings for them. If this is not possible, the reasons for this are reported and approximate query rewritings are used. The theory behind query answering can be found in [22] whereas a demo of the core components was presented at [23]. The module for automatically generating the sequence of changes among two ontology versions was presented at [4] whereas [24] and [25] report on other modules that try to respond to massive number of queries that might need to be changed by producing possible rewritings as well.

In order to evaluate the algorithms reported in this paper, we used a workstation with an Intel Core i7 processor running at 3.4 Ghz, and 4GB memory, using Windows 7x64. Moreover, we used two well-known ontologies: One medium-size ontology (CIDOC-CRM [6]) from the cultural domain which is rarely changed and one large-size ontology (Gene Ontology [6]) from the bioinformatics domain which is heavily updated daily.

CIDOC-CRM is an ISO standard which consists of nearly 80 classes and 250 properties. For our experiments we used versions dated from 02.2002 (v3.2.1) to 06.2005 (v4.2). The detected change log that was automatically produced identified 726 total changes from v3.2.1 to v4.2. Gene Ontology (GO) on the other hand, is composed of about 28000 classes and 1350 property instances. GO is updated on a daily basis and for our experiments we used the change log from 25.11.08 to 26.05.09. The change log that was automatically produced contained 4175 changes.

### 5.1 Answering provenance queries

Next, we present experiments concerning the scalability of the algorithms for answering provenance queries. We measured the average execution time for computing answers to *how/when* and *extended-how* provenance queries. To do that we exhaustively queried for *how/when* and *extended-how* all resources in the latest ontology versions and the results are presented in Figure 6 and Fig. 7.

As shown in the figures, for both ontologies the average time to produce a change path increased linear to the number of changes we had to search. This is in line with the complexity of our algorithms as we presented previously. Moreover, the time to compute answers to *extended-how* provenance queries is greater than computing answers to *how/when* queries. This is reasonable since in the first case more triples are being added to the list of triples that we are looking for in the sequence of changes.

However, we can see that the overhead for searching the added triples in the change path has little impact in the total execution time since the dominant factor is the number of change operations. So, for CIDOC-CRM after 726 change operations we only need 275 msec in average to compute *how/when* provenance

whereas for *why* provenance we need 280 msec. On the other hand, for Gene Ontology after 4175 changes we need 4611 msec for *how/when* queries and 4967 msec for *extended-how* queries.

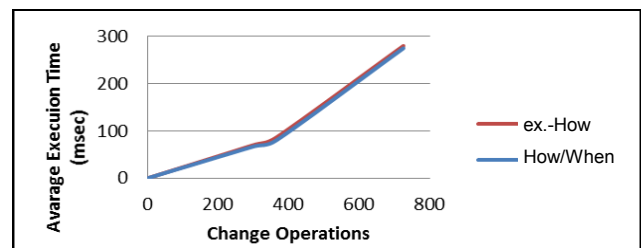


Figure 6. The average execution time compared to the number of changes for CIDOC-CRM

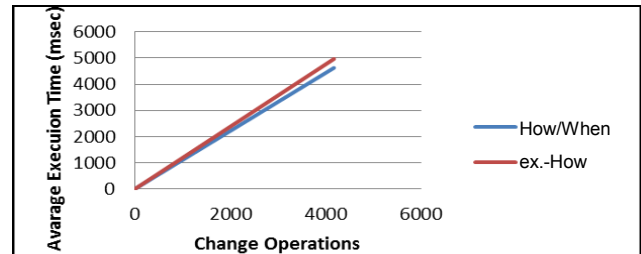


Fig. 7. The average execution time compared to the number of changes for Gene Ontology

Obviously, the time to compute a change path is greater for the Gene Ontology than for CIDOC-CRM. This is reasonable since for the Gene Ontology we have to search 4175 changes, whereas for CIDOC-CRM we only have to search 726 changes.

Moreover, we've identified that the biggest number of changes in a change path in the case of Gene Ontology was 8 whereas for CIDOC-CRM it was 5. So, independent of the number of changes between ontology versions the interested user needs to check at most 8 change operations (including change operations in comments) to understand how the specific part of the ontology has been evolved. We have to note here that the average number of change operations that a user had to examine was 2 for CIDOC-CRM and 4 for GO which shows the added value of our approach even for ontologies that change often.

Finally, trying to understand the provenance queries, we made several interesting observations. One of them for example, was the following: We identified that in the evolution of the CIDOC-CRM ontology from version v3.2.1 to version v3.3.2, one ontology engineer renamed the class "*E11 Modification*" to "*E11 Modification Event*". A few years later another ontology engineer was employed to evolve the ontology. So in v4.2 we can see that the class "*E11 Modification Event*" was again renamed to "*E11 Modification*". If the second ontology engineer had an indication of the previous renaming he would avoid cycles, he would be able to identify possibly the reasons behind each renaming since we are also able to show comments from the ontology evolution. So, using provenance queries to explore ontology evolution can be a valuable tool reducing greatly the time spent on understanding evolution.

## 6. CONCLUSION & DISCUSSION

In this paper, we argue that ontology evolution is a reality and that the problem of understanding ontology evolution is a fundamental problem in the area. Ontology engineers should have proper tools to help them understand the choices of the past. To that direction,

we presented a novel module that assists ontology evolution as the reality that ontology model changes.

Instead of just identifying and presenting the changes from one ontology version to another, our module can answer fine-grained provenance queries for a specific resource. It can identify *when* a resource was created, *how* it was introduced and it can present the change operations that lead to the creation (or deletion) of that resource and its evolution history. This greatly minimizes the total time for understanding ontology evolution. Experiments performed, show the potential impact of our approach. For example, for CIDOC-CRM provenance answers can be retrieved at most within 280 msec and for GO at most within 5sec even if there are more than 4000 changes that have to be examined. Moreover, ontology engineers have to examine at most 5 change operations for CIDOC-CRM and 8 change operations for GO to understand how the ontology evolved.

We need to note that we selected the specific language of changes for several reasons. One of them is because it is a high-level language of changes as already described in Section 2. Moreover, the language possesses nice properties such as *uniqueness*, *composition* and *inversion*. *Uniqueness* is a pre-requisite for our system whereas *composition* and *inversion* are desirable but not obligatory properties. So, instead of the specific language of changes other languages (and the corresponding detection algorithm) could be also used as long as they preserve *uniqueness*.

As future work, several challenging issues need to be further investigated. An interesting topic would be to extend our approach for OWL ontologies. Another interesting topic would be to present summaries of the evolved change path if they become too big. Ontology evolution is becoming more and more important topic and several challenging issues remain to be investigated in near future.

## 7. ACKNOWLEDGMENTS

This work was partially supported by the PlanetData NoE (FP7:ICT-2009.3.4, #257641), the eHealthMonitor (FP7-287509) and the MyHealthAvatar (FP7-600929) EU projects.

## 8. REFERENCES

- [1] Gruber, T.R. 1993. A translation approach to portable ontology specifications. *Knowl. Acquis.* 5, pp. 199-220.
- [2] Flouris, G., Manakanatas, D., Kondylakis, H., Plexousakis, D., Antoniou, G.: Ontology change, 2008. Classification and survey. *Knowl. Eng. Rev.* 23, pp. 117-152.
- [3] Plessers, P., Troyer, O.D., Casteleyn, S. 2007. Understanding ontology evolution: A change detection approach. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5, pp. 39-49.
- [4] Papavassiliou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V. 2013. High-Level Change Detection in RDF(S) KBs. *Transactions on Database Systems*, 38.
- [5] Doerr, M., Ore, C.-E., Stead, S. 2007. The CIDOC conceptual reference model: a new standard for knowledge sharing. *Tutorials, posters, panels and industrial contributions at the ER*, vol. 83, pp. 51-56.
- [6] Gene Ontology Consortium, 2004. The Gene Ontology (GO) database and informatics resource. *Nucl. Acids Res.* 32, D258-261.
- [7] Buneman, P., Khanna, S., Tan, W.C. 2001. Why and Where: A Characterization of Data Provenance, *ICDT*, pp. 316-330.
- [8] Benjelloun, O., Sarma, A.D., Halevy, A., Theobald, M., Widom, J. 2008. Databases with uncertainty and lineage. *The VLDB Journal*, 17, pp. 243-264.
- [9] Green, T.J., Karvounarakis, G., Tannen, V. 2007. Provenance semirings. *ACM SIGMOD-SIGACT-SIGART PODS*, pp. 31 - 40 ACM, Beijing, China
- [10] Chiticariu, L., Tan, W.-C. 2006. Debugging schema mappings with routes. *VLDB*, pp. 79-90.
- [11] Volkel, M., Winkler, W., Sure, Y., Kruk, S.R., Synak, M. 2005. Semversion: A versioning system for rdf and ontologies. *ESWC*.
- [12] Zeginis, D., Tzitzikas, Y., Christophides, V. 2007. On the Foundations of Computing Deltas Between RDF Models. *ISWC/ASWC*, pp. 637-651.
- [13] Noy, N.F., Chugh, A., Liu, W., Musen, M.A. 2006. A Framework for Ontology Evolution in Collaborative Environments *ISWC*, pp. 544-558.
- [14] Plessers, P., Troyer, O.D. 2005. Ontology Change Detection Using a Version Log. *ISWC*, pp. 578-592.
- [15] Rogozan, D., Paquette, G. 2005. Managing Ontology Changes on the Semantic Web. *IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 430-433.
- [16] Stojanovic, L. 2004. Methods and Tools for Ontology Evolution. *Phd.* Univ. of Karlsruhe.
- [17] Curino, C., Moon, H., Deutsch, A. and Zaniolo, C. 2013. Automating the database schema evolution process. *The VLDB Journal*, 22, 1, pp. 73-98.
- [18] Jim, E., Ruiz, N., Grau, B. C., Horrocks, I. and Berlanga, R. 2011. Supporting concurrent ontology development: Framework, algorithms and tool. *Data Knowl. Eng.*, 70, 1, pp. 146-164.
- [19] RDF Primer, W3C Recommendation: <http://www.w3.org/TR/rdf-primer/>
- [20] Theoharis, Y., 2007. On Graph Features of Semantic Web Schemas. *IEEE Transactions on Knowledge and Data Engineering*, 20, pp. 692-702.
- [21] Cali, A., Gottlob, G., Lukasiewicz, T. 2009. Datalog<sup>+</sup>: a unified approach to ontologies and integrity constraints. *ICDT*, pp. 14-30. ACM, St. Petersburg, Russia.
- [22] Kondylakis, H., Plexousakis, D. 2013. Ontology evolution without tears. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, 19, pp. 42-58.
- [23] Kondylakis, H., Plexousakis, D. 2011. Exelixis: Evolving Ontology-Based Data Integration System. *SIGMOD*, pp. 1283-1286.
- [24] Kondylakis, H., Plexousakis, D. 2011. Ontology Evolution in Data Integration: Query Rewriting to the Rescue. *ER*, pp. 393-401.
- [25] Kondylakis, H., Plexousakis, D. 2012. Ontology Evolution: Assisting Query Migration. *ER*, vol. 7532, pp. 331-344



# Skyline Ranking à la IR

George Valkanas  
Dept. of Informatics and  
Telecommunications  
University of Athens  
Athens, Greece  
gvalk@di.uoa.gr

Apostolos N. Papadopoulos  
Dept. of Informatics  
Aristotle University of  
Thessaloniki  
Thessaloniki, Greece  
papadopo@csd.auth.gr

Dimitrios Gunopulos  
Dept. of Informatics and  
Telecommunications  
University of Athens  
Athens, Greece  
dg@di.uoa.gr

## ABSTRACT

Skyline queries have emerged as an expressive and informative tool, with minimal user input and thus, they have gained widespread attention. However, previous research works tackle the problem from an efficiency standpoint, i.e., returning the skyline as fast as possible, leaving it to the user to manually inspect the entire skyline result. Clearly, this is impractical, even with a few dozen points. The techniques addressing this issue are computationally expensive, mapping to NP-Hard problems or having exponential complexity  $O(2^d)$  with respect to data dimensionality  $d$ . Moreover, the result is a *set*, lacking any quality-based ranking. In this paper, we propose a novel IR-style ranking mechanism for skyline points, based on the renowned *tf-idf* weighting scheme. We present efficient algorithms to compute the quality of a skyline point according to our technique, and induce a total ordering of the skyline set. Finally, we empirically evaluate the efficiency of our method with real-life and synthetic data sets.

## Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Search process; H.2.8 [Database Applications]: Data Mining; H.2.4 [Systems]: Query Processing

## Keywords

Skyline, Ranking, TF-IDF, Top- $k$

## 1. INTRODUCTION

Skyline queries were initially proposed in the context of databases in [2]. Their ability to empower multi-criteria decision making, with minimum user input, and their applicability in a series of domains has gained them considerable attention from the database community. Assuming *w.l.o.g.* that smaller values are preferred, we say that point  $p = (p.x_1, p.x_2, \dots, p.x_d) \in \mathcal{D}$  *dominates* point  $q = (q.x_1, q.x_2, \dots, q.x_d) \in \mathcal{D}$  (and write  $p \prec q$ ), when:  $\forall i \in \{1, \dots, d\}, p.x_i \leq q.x_i \wedge \exists j \in \{1, \dots, d\} : p.x_j < q.x_j$ . Simply put,  $p$  dominates  $q$ , if  $p$  is at least as good as  $q$  in every dimension, and it is strictly better than  $q$  in at least one. The skyline of  $\mathcal{D}$ ,

denoted as  $\mathcal{S}$ , is composed of all  $d$ -dimensional points that are not dominated by any other point.

**Related Work and Motivation.** Skyline queries are a well studied problem in the area of Computational Geometry [6], but have attracted considerable attention in the context of databases, when Börzsönyi et al introduced the *skyline operator* [2].

Several algorithms have been presented for skyline computation, with BBS [9] being the most preferred when using an index, due to its progressiveness and I/O optimality. Efficient algorithms have also been proposed in [5] and [10] for such cases where indexing cannot be applied. These works (as many more) focus on efficiency, i.e., retrieving the skyline as quickly as possible, and the result is a *set*, i.e. all points are equally important. In other words, there is no discrimination between the points, leaving it entirely to the user to select. Unfortunately, the skyline may contain far too many points: the skyline of randomly generated points is  $\Theta(\log^{d-1}(n))$  [1]. In an era when “ten blue links” seem too many [7], returning approximately  $10^3$  skyline points from a dataset with  $10^9$  points, immediately negates the advantages of skyline queries.

To address the skyline cardinality explosion problem, the general direction is to return a subset of  $k$  skyline points, where  $k$  is a user- or application-defined parameter. The subset has some specific properties, e.g., collectively maximizes coverage [8], captures the contour of the skyline [11], diversifies the skyline [12, 11], etc. Nevertheless, these techniques fail to differentiate between the returned points based on some importance measure. Moreover, they are mapped to NP-Hard problems, so we can only efficiently approximate the solutions, unless P=NP.

Researchers have also investigated ranking of skyline points. We identify two categories, depending on the amount of information used to rank the points: In the first case, the entire dataset is used, and the importance of a skyline point is given by the number of points it dominates [9, 14]. The major shortcoming of the first category is that all dominated points are equally important. For example, a point  $p_1$  dominated by 10 skyline points and another one  $p_2$  dominated by 100 contribute the same weight to their dominators. Taking into account that skyline queries have an inherent relation to sorting [1], and that distance measures for sorted lists heavily rely on the relative positions of items, it feels counter-intuitive to use the same weight for all dominated points.

The second category relies on the skyline  $\mathcal{S}$  alone, and typically uses dominance relations in subspaces [13, 4]. Consequently, such techniques ignore dataset characteristics. They are also generally inefficient, as they need to consider  $O(2^d)$  non-empty subspaces. Moreover, they favor skyline points with extreme values in a single dimension and have been shown to produce correlated results, whereas some of them [13] have not been sufficiently evaluated.

**Contributions.** To address these shortcomings, we present a novel

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

ranking scheme for skyline points. We argue that a point’s importance should be inversely proportional to the number of skyline points that dominate it. Additionally, our model distinguishes dominated points, based on their relative positions. For instance, if  $sp_1 \prec a$ ,  $sp_1 \prec b$ , and  $a$  and  $b$  do not dominate each other, they contribute equally to  $sp_1$ . Otherwise, if  $a \prec b$ , then  $score(a) > score(b)$ . Therefore, our technique promotes skyline points that dominate *genuine* points, i.e., points which are not dominated by many others, and is a hybrid approach.

To capture both aspects in a single scoring function, we apply a modified version of the renowned *tf-idf* weighting scheme and present efficient algorithms that rank the skyline according to this scheme. Our contributions are briefly described as follows:

- We define a novel, generic and intuitive measure of importance for skyline points. Inspired by the renowned *tf-idf* weighting scheme from information retrieval, our method promotes skyline points that dominate *genuine* points.
- We present efficient algorithms that compute the top- $k$  most important skyline points, given our measure.
- We provide an extensive experimental evaluation, in terms of efficiency using both real-life and synthetic datasets.

**Roadmap.** The rest of the paper is organized as follows. Section 2 gives the details of our scoring model and the algorithms proposed for the task at hand. In Section 4 we evaluate our techniques. Finally, Section 5 summarizes our findings and concludes the paper by discussing briefly future work in the area.

## 2. DP-IDP WEIGHTING SCHEME

Our proposed measure, *dp-idp*, which stands for *dominance power - inverse dominance power*, is inspired by the renowned *tf-idf* weighting scheme from Information Retrieval. The general rationale is that dominated points are not equally important, and that they impact skyline point differently. Therefore, their contribution depends on some local (per skyline point) and some global characteristics (the entire skyline), much like *tf-idf* uses local and global information to find important keywords in a document corpus. In the following paragraphs we present our ranking scheme.

### 2.1 Inverse Dominance Power

We will start with *inverse dominance power* (*idp*), which is easier to define, due to its more global view. The *inverse dominance power* of a point  $p \in (\mathcal{D} \setminus \mathcal{S})$  is the number of skyline points which dominate  $p$ . This factor is similar to *idf* in the sense that the more frequently  $p$  appears in a skyline point’s dominated set, the lower the importance of  $p$ . More formally:

$$idp(p) = \log \frac{|\mathcal{S}|}{|\{sp \in \mathcal{S} : sp \prec p\}|}$$

An interesting property of *idp* is the following: Assume a set of points  $q_1, q_2, \dots, q_m$  dominated by all skyline points, i.e.,  $\forall sp \in \mathcal{S}, sp \prec q_j, j = 1, \dots, m$ . The contributing score of the  $q_i$ ’s will be 0, due to the *log* in the *idp* factor. Such points do not alter the ranking of  $\mathcal{S}$ , either with ours or simpler models (e.g.,  $|\Gamma(sp)|$ ), because they affect all skyline points the same.

### 2.2 Dominance Power

There are several ways we could define the *dominance power* of a dominated point. Given that we want to measure this factor with respect to a skyline point  $sp$ , we argue that its relative position

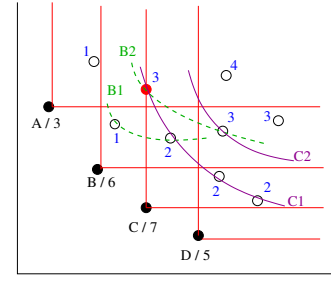


Figure 1: Example for the Dominance Power

to  $sp$  should matter. As a result, the same dominated point may contribute differently to different skyline points.

To avoid introducing more artifacts in our model, we choose the dominance relation as our building block. More specifically, we find the *layer of minima*<sup>1</sup>  $lm(p, sp)$  where the dominated point  $p$  falls in, with respect to  $sp$ . The dominance power of that point is then given by the inverse of the layer where it lies, i.e.,

$$dp(p, sp) = \frac{1}{lm(p, sp)}$$

Figure 1 portrays the skyline of a dataset, and the dominance region for each skyline point. Moreover, it shows the layers of minima for skyline points  $B$  and  $C$ , in dotted green and purple respectively. We observe that the red-filled point, dominated by both  $B$  and  $C$ , lies at different minima layers. Being easier to reach it from  $C$ , should render it more important for  $C$ . On the contrary, there is an additional layer for skyline point  $B$  prior to reaching that point, that decreases its importance. This is similar to *term frequency*, where the same term is weighted differently, depending on its occurrence in each document.

### 2.3 Putting it all together

Given our previous discussion, we can now formally introduce how we compute the importance of a skyline point  $sp$ . We use an additive model, because *i*) it is monotonous (dominating more points increases the overall importance) *ii*) it is comprehensive and *iii*) it leads to efficient computations, as we followingly discuss. Therefore, the importance of a skyline point  $sp$  is given by:

$$score(sp) = \sum_{p: sp \prec p} \frac{1}{lm(p, sp)} \times \log \frac{|\mathcal{S}|}{|\{sp' \in \mathcal{S} : sp' \prec p\}|}$$

The additive model also favors skyline points that dominate more *genuine* points, i.e., points dominated by few others in general (not just skyline points). This is important, because those skyline points are the reason why the dominated ones cannot be part of the skyline. For example, if we remove  $B$  from the skyline in Figure 1 (e.g., a hotel being fully booked), the point next to it will enter the skyline at once (similarly for the closest point dominated by  $A$ ). Additionally, points dominated by the entire skyline still have no effect. Finally, note that the sum of *tf - idf* values is also used in IR systems, to score the entire document against a query.

## 3. RANKING THE SKYLINE

Algorithm 1 gives the baseline approach to rank the skyline  $\mathcal{S}$  with our proposed scheme. For each skyline point  $sp$  (line 1),

<sup>1</sup>In the literature, the term *layer of maxima* is more common. Here, we use the term *layer of minima* because we assume that small values are preferable.

---

**Algorithm 1** Baseline
 

---

**Input:** Skyline  $\mathcal{S}$ , Dataset  $\mathcal{D}$ , Integer  $k$ 
**Output:** Ranked List

```

1: for every  $sp \in \mathcal{S}$  do
2:    $score(sp) \leftarrow 0$ ;  $layer \leftarrow 1$ ;  $lm \leftarrow NextLayer(sp, \emptyset)$ ;
3:   while ( $lm \neq \emptyset$ ) do
4:     for every  $p \in lm$  do
5:        $score(sp) += \frac{1}{layer} \times \log \frac{|\mathcal{S}|}{|\{sp': sp' \prec p\}|}$ ;
6:      $lm \leftarrow NextLayer(sp_i, lm)$ ;  $layer++$ ;
7: Order by descending  $score(sp)$ ;
8: Return  $k$  highest skyline points;
  
```

---

we extract one-by-one its minimal layers (lines 2–6). *NextLayer* uses BBS [9] internally. For every point in each layer (line 4), we find how many in  $\mathcal{S}$  dominate it, and together with the layer’s index, we update the score of  $sp$  (line 5). After ordering the skyline in decreasing score order (line 7), we return the top- $k$  ranked items.

Unfortunately, this approach is computationally expensive, due to repeated evaluations. It also computes the score of all skyline points, despite our interest in the top- $k$  results. Finally, it lacks any notion of *progressiveness*, as we need to rank the entire skyline first. For all these reasons, we present an alternative approach, that relies on bounding the score of a skyline point.

### 3.1 Bounding the score

Bounding the score of a skyline point  $sp$  will help us reduce computations, by pruning away those that will not make it to the top- $k$  positions. To achieve this, we use the number of points that  $sp$  dominates,  $|\Gamma(sp)|$ . We can then derive lower and upper bounds of the score of a skyline point, as shown in the next paragraphs.

**Loose Bounds.** The simplest bounds consider each skyline point independently of the rest, and are derived as follows. A skyline point obtains its maximum score when *all* the points it dominates are in the same (first) layer, and they are *not* dominated by any other skyline point. In that case, the upper bound is:

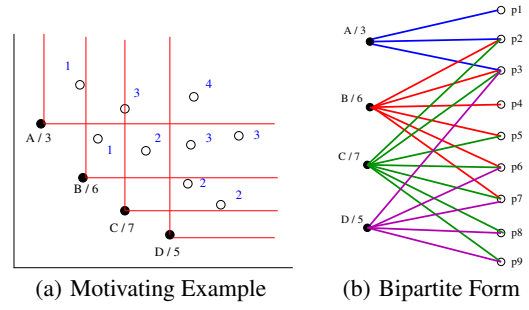
$$\overline{score}(sp) = |\Gamma(sp)| \times \log |\mathcal{S}|$$

On the other hand, the lower bound is obtained when every point is dominated by the entire skyline  $\mathcal{S}$ . In that case, the score is 0, due to the  $idp(sp)$  factor. However, this bound only holds for the skyline point  $sp_{min}$  with the minimum  $|\Gamma(sp_{min})|$ . The rest of the skyline dominates some points, which can not be dominated by  $sp_{min}$ . Consider, for instance, that  $|\Gamma(sp_{min})| = 3$  and that  $|\Gamma(sp')| = 5$ . By definition, the 2 additional points dominated by  $sp'$  can not be dominated by  $sp_{min}$ , otherwise  $|\Gamma(sp_{min})| = 5$ . Therefore, the surplus will be dominated by  $|\mathcal{S}| - 1$  skyline points, and a correlated distribution<sup>2</sup> will give the lowest score value. This gives a slightly better lower bound:

$$\underline{score}(sp) = \log \frac{|\mathcal{S}|}{|\mathcal{S}| - 1} \times \sum_1^{n - \min \Gamma} \frac{1}{i}$$

**Collaborative Bounds.** Despite their simplicity, the above bounds have limited pruning capability. Assume, for instance, a dataset  $\mathcal{D}$ , with  $|\mathcal{D}| = 1M$  and  $|\mathcal{S}| = 800$ . If  $|\Gamma(sp)| = 300K$ , then  $\overline{score}(sp) \simeq 871K$ , and  $\underline{score}(sp) \simeq 3 \times 10^{-3}$ . Note that a skyline point  $sp'$  with  $|\Gamma(sp')| = 1$ , has an upper bound of  $\sim 2.9$ , making it eligible for consideration in the second round! Apparently, the computational gains of such bounds are easily swept away.

<sup>2</sup>In a correlated distribution, each point is a minimal layer



**Figure 2: Example of skyline and bipartite domination graph**

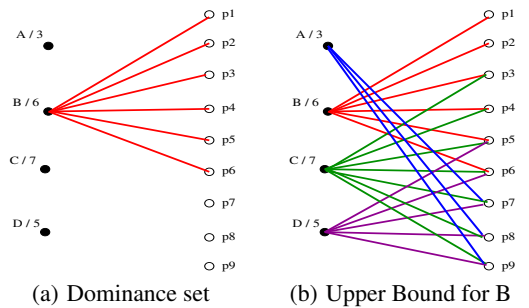
To address this issue, we derive stricter bounds, through additional information from other skyline points. To better understand this approach, we visualize the problem as a bipartite graph. Figure 2 shows a dataset, with its skyline and dominance regions on the left, and the resulting bipartite graph on the right. The left hand side of the graph contains the skyline, whereas the right hand side has the dominated points. We add an edge between a skyline point  $sp \in \mathcal{S}$  and a dominated point  $p \in \mathcal{D} \setminus \mathcal{S}$ , iff  $sp \prec p$ .

We start with the upper bound. Due to the additive model, the score of a skyline point is maximized when the contribution of each dominated point is maximized. It is easy to see that  $dp$  is maximized when the dominated point is at the earliest possible layer. To maximize  $idp$ , we rely on the Pigeonhole Principle. For any two skyline points  $sp_1, sp_2$ , if  $\Gamma(sp_1) + \Gamma(sp_2) > |\mathcal{D}|$ , then at least  $|\mathcal{D}| - (\Gamma(sp_1) + \Gamma(sp_2))$  dominated points are shared by  $sp_1$  and  $sp_2$ . Having more common points reduces  $idp()$ , so we only consider the minimum overlap. The question now becomes “*How should we assign the common edges to maximize the score of a skyline point?*”? Lemma 1 answers this question.

**LEMMA 1.** *Let  $sp$  be a skyline point,  $p_x$  and  $p_y$  two dominated points, where  $sp \prec p_x$  and  $sp \prec p_y$  and  $lm(p_x) = lm(p_y) = l$ . Assigning an edge to the point dominated by more skyline points gives a higher  $score(sp)$ .*

**PROOF.** Let  $\mathcal{S}_x, \mathcal{S}_y$  be the current sets of skyline points dominating  $p_x$  and  $p_y$ , respectively. Assigning an edge to either  $p_x$  or  $p_y$  gives two different bipartite graphs, with  $\mathcal{S}'_x$  and  $\mathcal{S}'_y$  being the new dominating sets of these points. It holds that  $|\mathcal{S}'_x| = |\mathcal{S}_x| + 1$  (same for  $\mathcal{S}'_y$ ), due to the new edge, i.e., one more dominating skyline point. The resulting bipartite graphs differ only in the assignment of this edge, which impacts the weights of  $p_x$  and  $p_y$ . The weights of all other dominated points remain unchanged. Assume that adding the edge to  $p_x$  yields a higher score. It so holds:

$$\frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}'_x|} + \frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}'_y|} > \frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}_x|} + \frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}'_y|} \Rightarrow$$



**Figure 3: Collaborative upper bound**

$$\log\left(\frac{|\mathcal{S}|}{|\mathcal{S}'_x|} \times \frac{|\mathcal{S}|}{|\mathcal{S}'_y|}\right) > \log\left(\frac{|\mathcal{S}|}{|\mathcal{S}_x|} \times \frac{|\mathcal{S}|}{|\mathcal{S}'_y|}\right) \Rightarrow |\mathcal{S}_x| \cdot |\mathcal{S}'_y| > |\mathcal{S}'_x| \cdot |\mathcal{S}_y| \Rightarrow$$

$$|\mathcal{S}_x| \cdot (|\mathcal{S}_y| + 1) > (|\mathcal{S}_x| + 1) \cdot |\mathcal{S}_y| \Rightarrow |\mathcal{S}_x| > |\mathcal{S}_y|$$

□

The above result tells us that a higher score is achieved by adding the extra edge to the dominated point with the higher indegree! Such a result can also be efficiently integrated in an algorithm to compute the upper bound of a skyline point's score. Figure 3(b) shows the edge assignment for the upper bound of skyline points  $B$ , using the above result.

A naive implementation of the upper bound can be very inefficient<sup>3</sup>, because it requires too many counter updates for the commonly dominated points. Since we must compute the bound of each skyline point  $sp$  independently and repeatedly, as new layers are extracted, we need a more efficient approach. Algorithm 2 presents this improved technique.

---

#### Algorithm 2 Score Upper Bounding

---

**Input:**  $\mathcal{D}, \mathcal{S}$ , Skyline Point  $poi$ ,  $minIDP$ ,  $layer$

**Output:** Upper bound of  $poi$

```

1:  $v_{idp}.push(minIDP)$ ;  $v_{pnd}.push(pending(poi))$ ;
2: Sort  $\mathcal{S}$ , in decreasing  $|\Gamma(sp)|$ ;
3: for every  $sp \in \mathcal{S}$ ,  $sp \neq poi$  do
4:    $surplus = |\Gamma(poi)| - seen(poi) + |\Gamma(sp)| > |\mathcal{D}|$ 
5:   if ( $surplus > 0$ ) then
6:      $v_{pnd}[last] = v_{pnd}[last] - surplus$ ;
7:      $v_{pnd}.push(surplus)$ ;
8:      $v_{idp}.push(v_{idp}[last] + 1)$ ;
9:  $ub \leftarrow 0$ ;
10: for ( $i = 0$ ;  $i < v_{idp}.size()$ ;  $i++$ ) do
11:    $ub \leftarrow \frac{1}{layer+1} * v_{pnd}[i] * \log \frac{|\mathcal{S}|}{v_{idp}[i]}$ 
12: Return  $ub$ ;
```

---

The improved algorithm takes as input the dataset  $\mathcal{D}$ , the skyline  $\mathcal{S}$ , the skyline point of interest  $poi$ , and two values  $minIDP$  and  $layer$ . As we extract more layers for  $poi$ , we must compute the number of skyline points dominating each of the extracted points, which we need for the  $idp$  value. The minimum value that we have seen this far is stored in  $minIDP$ , and is different for each skyline point. This value practically tells us that any point in subsequent layers will be dominated by *at least*  $minIDP$  skyline points, due to dominance being a transitive relation. The  $layer$  tells us which was the index of the last layer of minima extracted for  $poi$ .

The algorithm uses two vectors, storing the  $minIDP$  value and the number of *unseen* points, that have not yet been extracted for  $poi$  (line 1). For example, if  $|\Gamma(poi)| = 100$ , and we have extracted 30 points, then  $unseen(poi) = 70$ . The vectors practically store how many points ( $v_{PND}$ ) can be dominated by that many skyline points ( $v_{IDP}$ ). We sort the skyline points in decreasing order of their dominance power (line 2). We iterate over them (line 3), and select those points that will share common edges with  $poi$ , using the Pigeonhole Principle (lines 4–5). The surplus of points is removed from the last position (line 6) and is appended, incremented by 1 (lines 7–8). With these values, we can compute the upper bound according to the DP-IDP scheme (lines 10–11).

To better explain lines 5–7, assume  $v_{pnd}[last] = 60$ ,  $v_{idp}[last] = 4$ , and  $surplus = 25$ . This means that 60 points will be dominated by 4 skyline points and the current  $sp$  will share at least

<sup>3</sup>Our experiments showed that this step alone can make up for up to 10 seconds of CPU processing time.

25 dominated points with  $poi$ . As a result, we must add an edge (i.e., increment the  $idp$ ) for an equal number of unseen points from  $poi$ . These must be selected from the points with maximum current  $idp$ , due to Lemma 1. Processing the skyline in decreasing order of dominance power ensures that we are properly assigning edges, and that the maximum  $idp$  is in the last position. Given these values, 25 points will be computed with an  $idp$  of 5, which we append, whereas 60-25=35 will remain with an  $idp$  of 4, which we update.

For the lower bounds we could follow a similar reasoning. Unfortunately, the edge assignment problem in this case is not as easy. Although certain properties are self-evident, e.g.,  $dp$  decreases with a correlated distribution, they do not necessarily result in the lowest possible score for a point. Consequently, we may have to reassign edges, and, possibly, reconsider the layer where some points are (i.e., break the correlated distribution). Therefore, in our current work, we will not pursue the collaborative lower bounds further, but plan to actively investigate it in our ongoing work.

### 3.2 Skyline Ranking with IR-style

Now that we have shown how we can efficiently bound the score of a skyline point, using easily extracted information, we turn our focus to finding the top- $k$  most important skyline information, according to our DP-IDP weighting scheme. Algorithm 3 shows the general idea of execution of our technique, to efficiently compute the top- $k$  skyline points. Our algorithm processes the points according to a prioritization scheme, and employs pruning of skyline points that will certainly not be in the final top- $k$  result.

---

#### Algorithm 3 SkyIR

---

**Input:** Skyline  $\mathcal{S}$ , Dataset  $\mathcal{D}$ , Integer  $k$

**Output:** Top- $k$  list

```

1: for every  $sp \in \mathcal{S}$  do
2:    $sp_{\Gamma} \leftarrow |\Gamma(sp)|$ 
3:    $sp_{score} \leftarrow 0$ ;
4:    $priorityQueue.enqueue(sp_{prior}, sp)$ ;
5:  $kScore \leftarrow 0$ ;
6: while ( $!priorityQueue.empty()$ ) do
7:    $poi \leftarrow priorityQueue.dequeue()$ ;
8:   if ( $UpperBound(poi) < kScore$ ) then
9:     Discard  $poi$ ;
10:    continue;
11:   if ( $pending(poi) > 0$ ) then
12:      $lm \leftarrow NextLayer(poi, lm)$ ;
13:      $poi_{score} \leftarrow updateScore(poi, lm)$ ;
14:      $added \leftarrow topk.insert(poi, poi_{score})$ ;
15:     if ( $!added$  AND  $pending(poi) == 0$ ) then
16:       Discard  $poi$ ;
17:       continue;
18:     if ( $topk[k] > kScore$ ) then
19:        $kScore \leftarrow topk[k]$ 
20:   if ( $pending(poi) > 0$ ) then
21:      $priorityQueue.enqueue(sp_{prior}, sp)$ ;
22: Return  $topk$ ;
```

---

The algorithm starts by initializing appropriate information on the skyline points (lines 1–4), such as their dominance count, known score, and priority value, according to the prioritization scheme that we use (see below). We add each skyline point to a priority queue, using its priority value (line 4). We also initialize the  $k$ -th value, i.e., the value of the  $k$ -th ranked skyline point, to 0. We then enter a loop, each time extracting the top-most item from the queue  $poi$

(line 7). If the upper bound of that point’s score is below the  $k$ -th value, there is no need for further examination (lines 8 – 10). So we discard it and proceed with the next one from the priority queue. Otherwise, we extract the next layer of  $poi$ , provided there is one (lines 11–12). We update the point’s score using this layer (line 13) and try to add  $poi$  in the top- $k$  result. If the point was not added, and it can not be further updated, we discard it and proceed with the next point from the priority queue (lines 14–17). If the point was added, we keep track of the  $k$ -th value in the top- $k$  result. If we can further update it, we compute its new priority and add it back in the priority queue (lines 20–21). The loop ends when the priority queue becomes empty, meaning no other points can update their score. The top- $k$  list contains the final result.

**Priority Schemes.** Our SkyIR algorithm relies on a prioritization scheme to process the skyline points. In our current work we experiment with the following prioritization schemes.

- **Round Robin (RRB):** Items are processed in a round robin fashion. According to this scheme, we can not process the same skyline point twice, unless we have processed every skyline point first. This scheme also allows for an implementation that relies on arrays rather than the general priority queue, leading to faster (main memory) accesses.
- **Pending (PND):** The priority of an item is the number of points that it has not yet processed. For example, if a skyline point dominates 100 points, and it has already “seen” 30, its priority will be 70. Therefore, the more dominated points it has yet to see, the higher the priority of the skyline point.
- **Upper Bound (UBS):** The priority of a skyline point is given by the upper bound of its score. In other words, its priority is its potential to achieve a high final score. Similarly to the previous scheme, a higher upper bound results in a higher priority for the skyline point. Given that the upper bound can be used as a point’s priority, it is even more important to have an efficient technique to compute it, like Algorithm 2.

## 4. PERFORMANCE EVALUATION

In this section, we report on the results of our experimental evaluation. The experiments were run on a Quad-Core @2.5GHz machine, with 8Gb RAM, running Linux. The code was written in C++ and compiled with g++ 4.7.2, with -O3 optimization. The datasets we consider were indexed by an aggregate R\*-tree, with a 4Kb page size. An associated cache with 20% of the corresponding R\*-tree’s blocks was used with every experiment. Unless stated otherwise, the reported timings are in seconds, measured as CPU processing time and assuming a default value of 8ms per page fault.

**Datasets and Algorithms.** We generated datasets with *independent* (IND) and *anticorrelated* (ANT) distributions, as in [2], and also use Forest Cover<sup>4</sup>. Table 1 shows their basic properties. Although the datasets may seem rather small in size (up to 500K), one should keep in mind that our weighting scheme extracts *all* of the minimal layers for each skyline point. This problem is known to be difficult for high dimensionality even in the RAM model [3].

The algorithms that we evaluate are Baseline and SkyIR. For SkyIR we want to compare the performance of the **Loose** (LS) and **Collaborative** (CB) bounds, and how the three prioritization schemes affect the results. We use the abbreviations as suffixes to indicate what we compare each time.

**Runtime.** Figure 4(a) shows the total runtime for the independent distribution, when varying the dataset cardinality, with  $k=5$ .

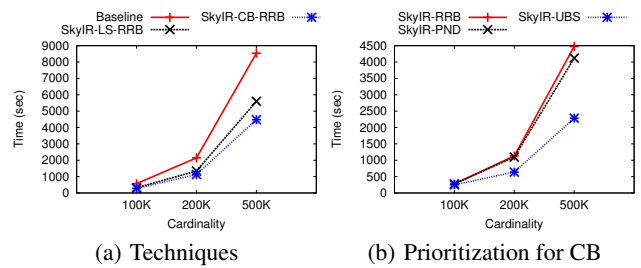
<sup>4</sup><http://kdd.ics.uci.edu>

**Table 1: Dataset Statistics**

Data set	Cardinality	Dimensionality
Independent (IND)	100K, 200K, 500K	2,3,4
Anticorrelated (ANT)	100K, 200K, 500K	2,3,4
Forest Cover (FC)	580K	2,3,4

The naive approach is the worst, whereas SkyIR with collaborative bounds performs the best of the techniques, and we have obtained similar results when varying dimensionality and  $k$ .

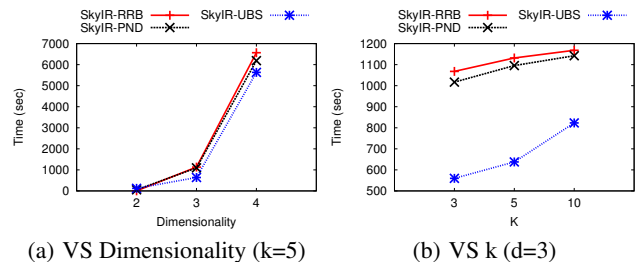
As seen in Figure 4(b), the UBS prioritization scheme outperforms all others, resulting in up to  $3\times$  improvement compared to the baseline. Similar results are obtained for different priorities with the loose bounds, but the differences are less pronounced. An important observation from these plots is that the problem we are solving is not linear with the cardinality of points. The reason is that as the cardinality increases, there are more minimal layers to extract, and the computational costs are increased a lot, as a result of both more CPU processing and page faults.



**Figure 4: Total runtime versus cardinality for IND,  $k=5$**

Figures 5(a)–(b) demonstrate how each prioritization scheme performs with the collaborative bounds. Figure 5(a) shows the performance when varying the data dimensionality. We observe that UBS performs the best for  $d = 3, 4$ , while being slightly worse for  $d=2$ . The reason for that is our array-based implementation, which is faster than the reordering of the priority queue maintained by PND and UBS. However, as seen in Figure 5(b) there is a huge improvement with UBS for  $d>2$ . The improvement increases with lower values of  $k$ , going up to 40%, because the collaborative bounds can prune away more points, reducing the computational costs.

Figures 6(a) and (b) demonstrate how the prioritization schemes perform for the ANT, versus dimensionality and  $k$ , respectively. Once again, UBS is better than PND. The loose bounds appear to be slightly better than the collaborative, but not considerably. The difference comes from the fact that the loose bounds are less computationally intensive. The more interesting fact, however, is that ANT appears to be *easier* when compared to IND. In particular, for  $d=4$ , it takes  $\sim 6000$  seconds for CB to compute the top-5 for IND, whereas it takes  $\sim 4000$  seconds for ANT. The reason is again that IND has more layers to extract, and is more CPU hungry. Even though ANT has a lot of page faults, its CPU time is minimal.



**Figure 5: Total runtime for various prioritization with IND, CB**



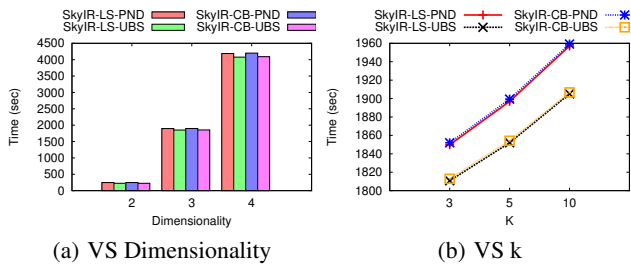


Figure 6: Total runtime for ANT distribution

Finally, Figure 7 compares the loose and collaborative bounds, when applying the UBS technique on the real dataset FC. We observe that the CB technique performs better than LB for all tested dimensions.

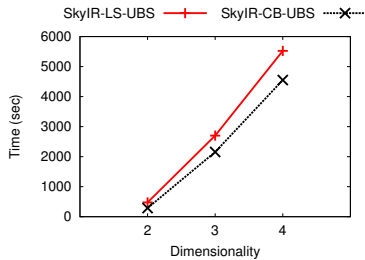


Figure 7: Forest cover

**Memory Consumption.** Finally, we compute the maximum number of items that we must maintain in memory while computing the top- $k$  result. Figures 8(a) and (b) show this for IND and ANT, respectively, using the CB technique. We observe that the number of maintained items increases as the cardinality of IND also increases. On the other hand, increasing the dimensionality of ANT, does not have a similar effect: the number of memory points increases as we go from 2D to 3D, but drops again as we proceed to 4D. This may be explained again by the fact that ANT has less layers spread the points more, increasing the points retrieved with each layer. This decreases the information we must store to proceed to the next layer, giving as the plot of Figure 8(b).

Generally speaking, the schemes RRB and UBS behave almost the same (with the exception of ANT). We should stress the fact, however, that the pending scheme (PND) *always* results in less memory utilization. This is because the scheme will stick to a single point and try to reduce its number of pending points as much as possible, whereas the other schemes will rotate more over different points. This is an interesting outcome, because PND would be a good alternative in systems with limited resources.

## 5. CONCLUSIONS

In this paper, we proposed a novel model for ranking skyline points, based on the renowned *tf-idf* weighting scheme from Information Retrieval domain. We presented efficient techniques for finding the top- $k$  result, by bounding the maximum score of a skyline point and employing pruning, combined with different visiting orders. We also experimentally evaluated the proposed bounding and prioritization schemes in terms of efficiency.

Since the layers of minima problem is a difficult one (especially for the external memory model) as future work we plan to investigate the alternative to check only the first few layers of minima instead of computing the whole set. This is expected to improve performance at the cost of result accuracy. A second direction is to

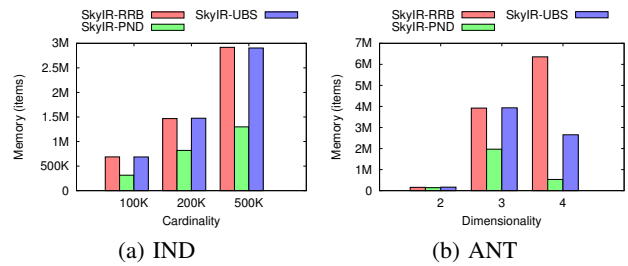


Figure 8: Maximum memory consumption for CB,  $k=5$

study the application of collaborative lower bounds in combination with the upper bounds studied in this work. Finally, we will work on the theoretical aspects of the problem in order to provide closed-form formulae for the cost of our algorithms. A potential starting point could be the application of a recurrence equation to estimate the number of layers combined with the cost of the BBS algorithm.

**Acknowledgements:** This work has been co-financed by EU and Greek National funds of the NSRF - Research Funding Programs: Heraclitus II fellowship, THALIS - GeomComp, THALIS - DISFER, ARISTEIA - MMD" and the EU funded project INSIGHT.

## 6. REFERENCES

- [1] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, Oct. 1978.
- [2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [3] A. L. Buchsbaum and M. T. Goodrich. Three-dimensional layers of maxima. *Algorithmica*, 39:275–286, July 2004.
- [4] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.
- [5] A. Das Sarma, A. Lall, D. Nanongkai, and J. Xu. Randomized multi-pass streaming skyline algorithms. *Proc. VLDB Endow.*, 2(1):85–96, Aug. 2009.
- [6] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22:469–476, 1975.
- [7] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. *ACM Trans. Inf. Syst.*, 19(3):242–262, 2001.
- [8] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The  $k$  most representative skyline operator. In *ICDE*, pages 86–95, 2007.
- [9] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
- [10] C. Sheng and Y. Tao. On finding skylines in external memory. In *PODS*, pages 107–116, 2011.
- [11] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.
- [12] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos. Skydiver: A framework for efficient skyline diversification. In *EDBT*, pages 406–417, 2013.
- [13] A. Vlachou and M. Vazirgiannis. Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships. *Data Knowl. Eng.*, 69(9):943–964, 2010.
- [14] M. L. Yiu and N. Mamoulis. Multi-dimensional top- $k$  dominating queries. *VLDB J.*, 18(3):695–718, 2009.

# Multi-Engine Search and Language Translation

Steven J. Simske  
Hewlett-Packard Labs  
3404 E. Harmony Rd. MS 36  
Fort Collins CO 80528 USA  
+1 970 898 1359  
Simske@hp.com

Igor M. Boyko  
Cisco Inc.  
Cisco Bldg 8, 3750 Zanker Rd  
San Jose CA 95134 USA  
+1 650 892 9924  
lgboyko@cisco.com

Georgia Koutrika  
Hewlett-Packard Labs  
1501 Page Mill Rd., MS 1157  
Palo Alto CA 94304 USA  
+1 650 857 2181  
Koutrika@hp.com

## ABSTRACT

Two of the most important elements in user interaction with a database are search and language translation. Search is used to access a database system through queries, for which the accuracy and completeness of response are key challenges. Language translation re-purposes content for a different audience, and the accuracy of translated text can be directly evaluated using search output similarity. In this paper, we summarize previously unpublished approaches to improving the quality of both search and translation, with an aim of improving the accuracy of both of these tasks. Specifically, multi-engine and related meta-algorithmic approaches are shown to be promising means of improving the performance of both search and translation. We then describe the vision of how search and translation can be combined to create a more robust overall text mining project.

## Categories and Subject Descriptors

G.2.1 [Mathematics of Computing]: Discrete Mathematics – *combinatorics*. G.4 [Mathematics of Computing]: Mathematical Software. H.3.3 [Information Systems]: Information Storage and Retrieval – *information search and retrieval*. I.2.7 [Computing Methodologies]: Artificial Intelligence – *natural language processing*.

## General Terms

Algorithms, Experimentation, Languages

## Keywords

Expert Feedback, Synonym, Meta-Algorithmics, Meta-Analytics, Search, Language Translation

## 1. INTRODUCTION

Automated search has been a research challenge of high interest to the data mining, knowledge generation and machine intelligence communities for the past several decades. Search queries are comprised of individual textual terms or multiple text terms associated with each other through, for example, a Boolean expression. These queries are often unreliable methods of obtaining the optimal set of documents (or other logical elements) from a corpus. This is due in part to the fact that no default search engine response to a query will provide a customized set of

documents matching what a particular user desired in entering the search query. In this paper, we review previous (and unpublished) work on providing the means to (a) extend the search capabilities of a search engine by increasing the likelihood that related documents are found for a particular search query; (b) increase the search efficacy when the user has only a vague idea about what she is trying to find; and (c) provide a means to optimize for several factors how to select documents associated with a query within any corpus. The methods used were part of the body of research to underpin the concepts in a recent book describing meta-algorithmics [1], but were not incorporated into the book.

Meta-algorithmics are a series of approaches to intelligent system design that describe how to combine two or more algorithms or systems into a single system for machine intelligence. Search is a form of machine intelligence associated with filtering; that is, narrowing down a larger body of data into a topic-specific body of data; that is, search output *information*. Language translation is a form of machine intelligence associated with conversion, or *transduction*, of one type of data into another.

The basis of the meta-algorithmic, multi-engine approach to search adopted in this paper was described earlier in patent application [2] which was not exercised. Thus, the research represents previous unpublished research with promising results suggestive of a useful future search research area. The meta-algorithmic approach considered is termed “synonymic search”, which allows a single search query to be expanded into a set of queries representing synonymic expressions for the original query. The approach also allows tuning of the amount of synonymic broadening to be applied to the received query for constructing the set of synonymic search queries. Identification of resulting documents responsive to each of the plurality of queries is received, and such received documents are ranked based at least in part on a weighting assigned to each of the plurality of queries.

Language translation, like search, is an important tool for data mining and knowledge generation. In this paper, we present a simple meta-algorithmic approach that combines the output of multiple language translations and uses “expert feedback” in the form of a dictionary in the target language of the translation.

## 2. SEARCH

Expanding a single search query into a series of related searches is known as query expansion. In this paper, query expansion is incarnated through the use of term synonyms. That is, each term in a query that has one or more synonyms triggers the expansion of the query into a set of parallel queries, each one including only one of this set of synonyms. This process is repeated for every one of  $N_S$  terms in the query having one or more synonyms, resulting in a total number of queries,  $N_Q$ , given by Equation 1, where  $S_i$  is the number of synonyms for term  $i$ .

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

$$N_Q = \prod_{i=1}^{N_s} (S_i + 1) \quad \text{Equation 1}$$

A set of synonymic queries is generated. Many commercially-available, freely-available and proprietary synonym lists exist. WordNet [3], for example, provides the means to generate such a list, and thesaurus options within many word processor engines provide the means to augment the list. Nouns, verbs and adjectives are the common parts of speech used for synonymic queries. In fact, many common articles (“the”, “a”, and “an”), prepositions (“of”, “with”, etc.) and conjunctions (“but”, “and”, and “or”, except when the latter two are used in Boolean searching capacity) are ignored altogether in most search engines. Many existing search engines, moreover, separate notions (idioms) consisting of two words into two separate terms, such as in the case of “take off” and “put up” (in which they are treated as “take” and “off” and “put” and “up”).

The synonymic search set is typically limited to proximate (and not associated) synonyms in order to keep the number of searches manageable, per Equation 1. Moreover, expressions such as “take off” and “put up” are treated as single candidates for synonyms, resulting in synonyms such as “launch” & “elevate”, or “erect” & “construct”, rather than synonyms for the individual words in these idioms. Further control over the total number of searches generated is obtained by limiting the number of proximate synonyms, denoted P, to an absolute maximum of, for example, five synonyms (P=5). If there are N terms for which synonyms are found in the original query, there are NP total searches possible. However, to prevent an open-ended number of queries, the total number of queries may be limited to an absolute maximum Q of, for example, 25 queries (most search engines are fast enough nowadays, at several hundredths of a second per query, that this value will typically limit the total search time to < 1 second of searching). The user may also be allowed to limit the total number of searches via a UI device.

Now, if  $NP > Q$ , the  $Q-1$  additional searches (the original query is always used) are pruned based on the relative synonymic relationship between each of the terms. An example illustrates this point. Suppose the user types in the query “class list for Stanford”. For the term “class”, the user will get the following synonyms: set, group, division, grade, rank, category, order (etc.). For the term “list”, the user will get the following synonyms: catalog, inventory, register, record, roll, directory (etc.). Already, the number of possible synonymic queries is 56 (that is,  $8 \times 7$ ), but no more than 25 are allowed (fortunately, “Stanford” is a relatively unique term – although “Stanford University” can be considered a synonym for it, this synonym does not expand the search, and so is ignored). The obvious solution is simply to accept 5 terms for “class” and 5 terms for “list”, but this is in general an unsatisfactory solution. Instead, the preferred implementation is to have the synonym database structured such that the synonyms are rated for their “closeness” or “proximity” to the original word. Let us suppose these rates for “class” are 0.9 (set), 0.85 (group), 0.72 (division), 0.65 (grade), 0.51 (rank), 0.42 (category) and 0.23 (order); and for “list” are 0.95 (catalog), 0.9 (inventory), 0.88 (register), 0.85 (record), 0.84 (roll) and 0.46 (directory). The highest 25 combinations are then found by multiplying the synonymic rates together, and so the highest ranking is for “class list Stanford” (1.0), followed by “class catalog Stanford” (0.95), continuing to #24 (“grade catalog

Stanford” at 0.6175) and #25 (“division record Stanford” at 0.612).

Note that the “weights” or “proximities” defined above can be further weighted/treated by the “semantics” of the query—i.e. if a query asks, as in the example below, for a “ball sport” then any synonyms of “ball” denoting “dancing” rather than “sports equipment” should be discarded. Such semantic weighting is, in general, quite difficult, and so weighted synonyms such as those demonstrated here help work around this problem. Note that the weights can be defined (a) manually; (b) automatically based on the co-occurrence of such terms in web sites, documents, corpuses, etc. – for instance, reference [4] has a statistical database generated from the British National Corpus, a 100 million word electronic databank sampled from the whole range of present-day English, spoken and written; and (c) automatically based on the order the synonyms occur in a linguistic engine such as WordNet. Almost the same statistical approach can be used for determining the parts of speech (POS) at the front end of query analysis. For example, the word “class” may be a noun, verb or adjective. Using the statistical results from [4], the word “class” is found to be most commonly typed as a noun, and so the appropriate noun synonyms can be used. If, however, a POS analysis of the query indicates that the word “class” is a verb, verb synonyms are found for “class”. This is also true of the word “list”, which can be both a noun and verb. Since even the best POS engines make mistakes, the user can be allowed to change the POS at the UI level, if available, if they think the engine may have misinterpreted the query.

It is clear from the above that there are numerous approaches to query expansion associated with synonymic proximity along with likely synonymic relevance of the term. After all the search queries have been defined, they are actually run on one or more search engines. On the internet, these search engines can be commercially available ones. On intranets and specific corpuses, they can be whatever search engine the user has available. In this step, all of the queries are provided as input for the search engine and the search engine returns the web sites, documents, etc. that it determines to be best matches. These matches are typically presented in order of relevance, utility, hit frequency, or other reasonable metric, and are presented to the user ranked from 1 to M, where M is the number of “hits” or “matching pages” found.

This approach, “by priority”, can use the following types of weighting to combine the search output of multiple engines (note that this is a separate weighting from the query weighting described above): (1) the engines themselves may be weighted by the confidence in the engines; and (2) the order of the results may be weighted, according to their rank in the output set provided by the search engine. It is worth noting, however, that even if a single search engine is used, the synonymic approach effectively provides a multi-engine output. Each is consistent with a meta-algorithmic Weighted Voting pattern [1]. A second means of presenting search output options to users is “by query”. This is simple, and has many possible incarnations. For example, each of the original and synonymic searches is presented as a link to the user, and the user can select any of them to find the highest priority sites presented. Another example is to present a tree of the original and synonymic searches [5]. These two approaches have different advantages. The “by priority” approach tends to smooth over biases of a search engine, providing averaging, while the “by query” approach provides quick alternative lists to the user. A preferred motif may be to present the results from the “by



priority” approach with links to the original and synonymic queries in an adjacent column.

An additional presentation mode is possible. In this mode, the overall relevance of all the search results is determined by comparing its keywords to those in the original query. For example, suppose the following two web page descriptions result: (a) a list of people suing Stanford for copyright infringement, and (b) a directory of classes in the Stanford biology program. The first search has “list” at 1.0, “Stanford” at 1.0 and no synonym for class. Its total synonymic weight (using the simplest weighting schema) is thus 2.0. The second search has “directory” for 0.46, “class” (lemma for classes) for 1.0, and “Stanford” for 1.0, for a total weighting of 2.46. Thus, the second search is deemed “more semantically similar” to the original query and is presented higher up in the results. This is the “by semantic weight” approach.

A real example is overviewed here. On one of the major internet search engines, the following query was entered: “ball sport in New Zealand” for which we were trying to find the name of a sport in which you get inside a large plastic double-walled ball and roll down a hill (called “zorbing”, a New Zealand invention) and the name for a sport similar to basketball played by women there (“netball”). Both are quite literally ball sports in New Zealand, but they are quite different from the set of top ten results that result for this query in most search engines (almost all are rugby, with basketball or volleyball occasionally making an appearance). The chief synonyms were sphere, globe & orb for ball; game, activity, team game & hobby for sport. The original search “ball sport New Zealand” found chiefly rugby sites, with some hockey and water sports interspersed in the top 10 priority sites. Ditto for “sphere sport New Zealand”. When the synonymic search “globe sport New Zealand” was performed, more water sports sites showed up. When “orb sport New Zealand” was queried, zorbing made its first appearance in the high priority list of sites. Water polo appeared when “ball activity New Zealand” was queried; croquet & volleyball when “ball team game New Zealand” was queried; and netball when “ball game New Zealand” was queried. This example illustrates the diversity of returns possible with the use of synonymic query.

### 3. LANGUAGE TRANSLATION

As a brief introduction to the use of multi-engine translators to increase overall translation accuracy, we used the meta-algorithmic pattern of Expert Feedback [1] where the “expert” was an English language dictionary [4] and the sources to be translated were either in Italian and Russian. Two 500-word documents were hand ground-truthed by the authors and three translation services were deployed (References [6], [7], [8] and [9] for the Italian-English translation).

**Table 1. Italian-English Translation**

Translator	1	2	3	Combined
Matching %	89.5	91.4	94.1	97.7

**Table 2. Russian-English Translation**

Translator	4	5	6	Combined
Matching %	80.5	84.9	93.4	96.2

The multi-engine approach for language translation used was straightforward. The words associated with the output of the multiple translations were directly aligned so that the terms could be matched directly for all three translations. Where the translation resulted in non-English words for one or more of the translators, the English word of another translator was used instead. If different English words were identified by the translations, then either the most commonly selected word or else the word provided by the engine with the greatest overall number of successes (English words) was used. This simple multi-engine scheme (Tables 1-2) resulted in reduction of the error rate by 61% (Italian) and 50% (Russian) in comparison to the error rate of the best single engine. Thus, as for search, a multi-engine approach to language translation showed considerable promise.

### 4. DISCUSSION

Several multi-engine approaches to search and language translation have been demonstrated in this paper. Synonymic and part-of-speech query expansions, in addition to a meta-algorithmic Weighted Voting approach, were shown to provide distinct advantages for customizing search output. Multi-engine alignment and best output acceptance was shown to significantly improve the quality of language translation for two short documents using two distinct languages. Obviously, further quantitative evaluation of the approaches outlined herein will be a useful next set of experiments. This paper only highlights the large set of possibilities in this space. In the future, validation of the techniques with IR datasets from TREC (queries and qrel ground truth) will be performed. The approaches outlined here will also be compared to relevant similar approaches, including query expansion, exploitation of synonyms and cross-language IR. Finally, it should be noted that there is a logical link between these two fields of text data filtering and transduction. Namely, the accuracy of the language translation approach can be directly gauged by comparing the search results on the un-translated and subsequently translated corpora. If the translation is accurate, then the documents should respond very similarly to un-translated and translated *queries* against the corpora. This type of *functional* testing of un-translated and translated corpora also warrants further, quantitative investigation. This will be a focus of future research for our team and, hopefully, others.

### 5. REFERENCES

- [1] Simske, S.J. 2013. *Meta-algorithmics: patterns for robust, low cost, high quality systems*. Wiley & Sons, Hoboken, NJ, USA, 386 pages.
- [2] Simske, S.J. and Boyko, I. 2002. System and method for management of synonymic searching. US Patent Application 10/256,674 (20040064447).
- [3] Word Net, <http://www.cogsci.princeton.edu/~wn/>.
- [4] British National Corpus, <http://www.natcorp.ox.ac.uk/>.
- [5] Vivisimo, <http://www.vivisimo.com>.
- [6] [http://inews.tecnet.it/show.asp?f=articoli/2002/04/IN0204\\_Focus\\_Kids.htm](http://inews.tecnet.it/show.asp?f=articoli/2002/04/IN0204_Focus_Kids.htm).
- [7] Free Translation, <http://www.freetranslation.com/>.
- [8] LinguaTec, <http://www.linguatec.net/online/>.
- [9] WorldLingo, <http://www.worldlingo.com/wl/Translate>.

# Querying Graph Structured Data (GraphQ)

Federica Mandreoli (University di Modena and Reggio Emilia, Italy)  
Riccardo Martoglia (University di Modena and Reggio Emilia, Italy)  
Wilma Penzo (University of Bologna, Italy)

# An Event-Driven Approach for Querying Graph-Structured Data Using Natural Language

Richard A. Frost, Wale Agboola, Eric Matthews and Jon Donais  
School of Computer Science  
University of Windsor, Canada  
richard@uwindsor.ca

## ABSTRACT

An ideal way for people to query graph-based knowledge, including triplestores in the semantic web, would be for them to ask questions in a natural language (NL). However, existing NL query interfaces to graph-based data have limited expressive power and cannot accommodate arbitrarily-nested quantification (i.e. phrases such as “a gangster who joined every gang”) together with multiple complex prepositional phrases, such as “in a city located in Illinois in 1918 using a set of keys that was stolen from a gangster”. It would appear that the commonly-used “entity-based” triplestores, together with what has become the de-facto approach of converting NL queries to SPARQL queries before being evaluated, hinders the development of expressive NL query processors. The reason is that entity-based triples are not conducive to the development of semantic theories of complex prepositional phrases, and the development of such theories is made considerably more complex when translation to SPARQL has to be taken into account. An alternative approach, which uses “event-based” triplestores, treats (bracketed) English queries as expressions of the lambda calculus which can be evaluated directly with respect to the triplestore. This approach facilitates the development of a formal denotational semantics of English queries which easily accommodates complex prepositional phrases. The approach described here could be used to develop a denotational semantics for a highly-expressive NL query language, and then that semantics could be used to guide the design of an NL query to SPARQL translator, thereby taking advantage of SPARQL optimizations.

## Categories and Subject Descriptors

H.2.4 [database management]: Query processing; H.5.2 [user interfaces]: Natural language

## General Terms

Theory

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

## 1. INTRODUCTION

The fact that Al Capone joined the Five Points Gang can be represented by the following triple:

```
(.../capone, .../joined, .../fpg)
```

where .../ are Uniform Resource Identifiers (URIs) for namespaces, and .../capone is a URI for a person. However, there is a problem with this approach. It is difficult to add related data such as the fact that Capone joined the FPG gang in 1914. It is insufficient to simply add the triple:

```
(.../capone,.../year_joined_gang, .../1914)
```

as this does not provide the necessary link between the two facts (the link is necessary because Capone joined several gangs). Various approaches are available to overcome this problem, only one of which concerns us in this paper. The approach in which we are interested is one which uses events rather than entities as subjects in the triples. For example, the fact that Al Capone joined the Five Points Gang can be represented as follows (note that from now on, we use ENT “capone” in place of “.../capone” etc.

```
{(EV 1001, REL "type", TYPE "join_ev"),  
(EV 1001, REL "subject", ENT "capone"),  
(EV 1001, REL "object", ENT "fpg")}
```

We can add the fact that Capone joined the Five Points Gang in 1914, with:

```
(EV 1001, REL "year", ENTNUM 1914)
```

A particular advantage of this approach is that the use of events facilitates the creation of a powerful denotational semantics for NL queries to graph-based data. In particular, use of events enables us to create an NL semantics with the following six properties: 1) the semantics is denotational in the sense that English words and phrases have a well-defined mathematical denotation (meaning), 2) the meaning of a composite phrase can be created by applying simple operations to the meanings of its components, 3) it is referentially transparent in the sense that the meaning of a word or phrase (after syntactic disambiguation) is the same no matter in what context it appears. 4) there is a one-to-one correspondence between the semantic rules describing how the meaning of a phrase is computed from its components and the syntactic rules describing the structure of the phrase, 5) it is computationally tractable, and 6) the meanings of words are defined directly in terms of primitive triple-store retrieval operations.

These six properties enable NL triple-store query processors to be implemented as highly modular syntax-directed interpreters. The advantage of this is that processors for individual language constructs can be built and tested separately. Consequently, the query processors can easily be extended to accommodate new language constructs such as prepositional phrases.

The semantics that we have developed is complex and is the result of two revisions that we have made to a well-known formal semantics of English, called Montague Semantics (MS) [6]. We first modified MS to create a computationally tractable form called FLMS which is suitable as a basis for NL query interfaces to relational databases. We then modified FLMS to a form which we call EV-FLMS which is suitable as a basis for querying event-based triple stores. Owing to the complexity of these modifications, this paper describes the two revisions in two separate sections.

The paper is structured as follows: in 2.1 we introduce Montague Semantics. In 2.2 and 2.3 we show how MS can be extended and converted to a computationally tractable form which can be used as a basis for NL query interfaces to conventional relational databases. In section 3 we discuss in more detail how knowledge can be represented in triple stores. In 4 we introduce some primitive retrieval operators for triple stores. In 5 we introduce a new event-based version of FLMS, called EV-FLMS and show how the meaning of words and phrases can be defined in terms of the triple-store retrieval operators. In 5.2 we show how the semantics accommodates prepositional phrases, thereby achieving all six of the properties listed above. In 6 we give examples of how complex queries such as “which gangster who stole a car in 1908 or 1918 in Manhattan joined a gang that was joined by Torrio?” are answered. In 7, we briefly discuss the use of a parser to disambiguate queries. In 8 we mention related work. We conclude in 9 with work yet to be done.

## 2. A COMPOSITIONAL SEMANTICS FOR NL RELATIONAL DB QUERIES

We begin by discussing MS and show how the meaning of a sentence in English can be composed from the meanings of its component words and phrases. We then show how MS can be extended and modified for use as a basis for NL query interfaces to conventional relational databases.

### 2.1 Montague Semantics

If we ignore that part of MS which deals with intensional and modal aspects of language, common nouns such as “thief” and intransitive verbs such as “smokes” can be thought of as denoting predicates over the set of entities in the “universe of discourse”, i.e. characteristic functions of type  $\text{entity} \rightarrow \text{bool}$ , where  $x \rightarrow y$  denotes the type of functions whose input is a value of type  $x$  and whose output is of type  $y$ . One of Montague’s many insights is that proper nouns (i.e. names) do not denote entities directly. Rather, they denote functions defined in terms of entities. For example, the proper noun “Capone” denotes the function  $\lambda p \text{ cap}$  where  $\text{cap}$  represents the entity associated with the name “Capone”. (For readers not familiar with the lambda calculus, the expression  $\lambda x \ e$  denotes a function which, when applied to an argument  $y$ , returns as result the expression  $e$  with all instances of  $x$  in it replaced by  $y$ .) According to the rules proposed by Montague, the phrase “Capone smokes”

(ignoring temporal aspects) is interpreted as shown below, where  $a \Rightarrow b$  indicates that  $b$  is the result of evaluating  $a$ ,  $\|x\|$  represents the denotation (meaning) of the word or phrase  $x$ , and  $x\_pred$  is the predicate associated with the word  $x$ .

```

||Capone smokes|| => ||Capone|| ||smokes||
=> ( $\lambda p \text{ p cap}$ ) smokes_pred
=> smokes_pred cap

```

Quantifiers such as “every”, and “a” denote higher-order functions of type:  $(\text{entity} \rightarrow \text{bool}) \rightarrow ((\text{entity} \rightarrow \text{bool}) \rightarrow \text{bool})$   
For example:

```

||every|| =  $\lambda p \lambda q \forall x (p \ x) \rightarrow (q \ x)$ 

```

where  $\rightarrow$  denotes logical implication in this context. Accordingly, the phrase “every thief smokes” is interpreted as:

```

( $\lambda p \lambda q \forall x (p \ x) \rightarrow (q \ x)$ ) thief_pred smokes_pred
=> ( $\lambda q \forall x \text{ thief\_pred } (x) \rightarrow q(x)$ ) smokes_pred
=>  $\forall x \text{ thief\_pred } (x) \rightarrow \text{smokes\_pred}(x)$ 

```

There are many advantages to MS including the fact that phrases of the same syntactic type, e.g. “Capone” and “every thief” have denotations of the same type i.e.  $(\text{entity} \rightarrow \text{bool}) \rightarrow \text{bool}$ , making the semantics highly compositional with respect to the syntactic structure of the phrase. Consequently, the semantics is easy to implement in a syntax directed interpreter as there is a one-to-one correspondence between the syntactic rules of the grammar defining the query language and the semantic rules defining how the meaning of a compound construct is computed from the meaning of its components). There are, however, two disadvantages of directly implementing MS as the basis for a database query processor, as discussed in the next two sub-sections.

### 2.2 An explicit denotation for transitive verbs

MS is not fully compositional as it does not have an explicit denotation for transitive verbs. Instead it leaves transitive verbs uninterpreted throughout the rewriting of the lambda expression denoting the sentence or phrase, and only deals with the transitive verb at the very end through a syntactic rewrite rule (see page 216 in [6] for the details). In earlier work [11] we developed a method for defining the denotation of transitive verbs explicitly. Accordingly, the denotation of “join” (as in “join a gang”) is as follows:

```

||join|| =  $\lambda z \ z(\lambda x \lambda y \text{ join\_pred } (y, \ x))$ 

```

Where  $\text{join\_pred}$  is the two place predicate corresponding to the word “join”. Note that this is similar to, but not exactly the same as, that proposed by Hendricks [18], Main and Benson [22], Blackburn and Bos [2] (who attribute it to Robin Cooper at the University of Goteborg), and Clifford [4]. The following shows the use of this denotation:

```

||Did Capone join the Five Points Gang?||
by parsing
=> ||Capone|| (||join|| ||Five Points Gang||)
=> ( $\lambda p \text{ p cap}$ ) (( $\lambda z \ z(\lambda x \lambda y \text{ join\_pred}(y, x))$ ) ( $\lambda q \text{ q fpg}$ ))
=> ( $\lambda p \text{ p cap}$ ) (( $\lambda q \text{ q fpg}$ ) ( $\lambda x \lambda y \text{ join\_pred}(y, x)$ ))
=> ( $\lambda p \text{ p cap}$ ) (( $\lambda x \lambda y \text{ join\_pred}(y, \ x)$ ) fpg)
=> ( $\lambda p \text{ p cap}$ ) ( $\lambda y \text{ join\_pred}(y, \ \text{fpg})$ )
=> ( $\lambda y \text{ join\_pred}(y, \ \text{fpg})$ ) cap
=> join_pred(cap, fpg)

```

which returns True if Capone joined the Five Points Gang.

### 2.3 An efficient version of MS

Another disadvantage of MS as a basis for database query processors is that a direct implementation of the denotations

of phrases which include the word “every” is computationally intractable. This is due to the fact that the function which denotes the word “every” requires all entities in the universe of discourse to be examined. For example:

$$\forall x \text{ thief\_pred}(x) \rightarrow \text{smokes\_pred}(x)$$

In order to overcome this problem, MS can be converted to a semantics called FLMS that is based on sets and relations rather than on their corresponding predicates. In this approach, which was first suggested and partially implemented by Frost and Launchbury [9] and further developed by Frost and Fortier, [15], sets and relations are used in the denotations of common nouns, intransitive verbs, and transitive verbs, rather than their corresponding predicates, and all other denotations are modified appropriately:

```

||thief|| = {capone, torrio, moran ..
||gang|| = {bowery, fpg ..
||smoke|| = {capone, torrio, moran ..
||Capone|| =  $\lambda p$  capone  $\in$  p
||Moran|| =  $\lambda p$  moran  $\in$  p
||Torrio|| =  $\lambda p$  torrio  $\in$  p
||Five Points Gang|| =  $\lambda p$  fpg  $\in$  p
||every|| =  $\lambda s \lambda t$  s  $\rightarrow$  t
||a|| =  $\lambda s \lambda t$  s  $\cap$  t  $\neq$  {}
||and|| =  $\lambda f \lambda g$   $\lambda s$  ((f s) & (g s))
||join|| =  $\lambda q$  {x|(x,image_x)  $\in$  collect(join_rel)
& q(image_x)}
join_rel = {(capone, bowery),(capone, fpg),
(torrio, fpg),etc.}

```

The definition of ||join|| uses relative set notation: informally, {a | b1  $\in$  s1 etc., & c1 etc.} is read as the set of all a such that b1 is a member of the set s1, etc. and c1 is a condition, etc. The function collect is defined such that it returns a new binary-relation, containing one binary tuple (x, image\_x) for each member of the projection of the left-hand column of join\_rel, where image\_x is the mathematical image of x under the relation join\_rel. For example:

```

collect join_rel
=> {(capone, {bowery,fpg}),(torrio, {fpg}), etc.

```

As example of the use of the denotation of the word “join”, consider: ||join|| ||Five Points Gang||

```

=>  $\lambda q$ {x|(x,image_x)  $\in$  collect(join_rel) & q(image_x)}
      ( $\lambda p$  fpg  $\in$  p)
=> {x|(x,image_x)  $\in$  collect(join_rel) &
      ( $\lambda p$  fpg  $\in$  p)(image_x)}
=> {x|(x,image_x)  $\in$  collect(join_rel) &
      (fpg  $\in$  image_x)}
=> {capone, torrio}

```

The resulting semantics is highly compositional: denotations of compound phrases and sentences are created using function application, according to the syntactic structure of the query. It should be noted that syntactic ambiguity is accommodated by having the parser generate more than one syntax tree each of which determines an order of application of the functions which are denoted by the words and phrases in the query. For example: one of the two syntactic parses of the query “Did Capone and Torrio join a gang?” would result in the following expression, which has the same meaning as the query “Did Capone join a gang and did Torrio join a (not necessarily the same) gang?”

```

(||and|| ||Capone|| ||Torrio||) (||join|| (||a|| ||gang||) )

```

which evaluates to True w.r.t. the definitions given above. We discuss ambiguity further in section 7.

The set-based FLMS semantics has the first five of the six properties discussed earlier. It can be implemented directly as part of a syntax-directed query processor for conventional relational databases in any programming language, but most easily in languages such as LISP, Miranda, Haskell, Scheme, ML or Python which support higher-order functions. Denotations of common nouns such as “thief” and intransitive verbs such as “smokes” are defined directly in terms of unary relations in the database. Relations such as join\_rel, which are used in the denotations of transitive verbs, are defined directly in terms of binary-relations.

### 3. EVENT-BASED TRIPLE STORES

Before we discuss how to convert FLMS to a form that can be used with event-based triple stores, it is helpful to consider further how knowledge can be represented in such stores. First, we consider how to represent facts associated with intransitive verbs. For example, the fact that Capone was known to smoke. This can be represented as:

```

{(EV 1005, REL "type", TYPE "smoke_ev"),
 (EV 1005, REL "subject", ENT "capone")}

```

Next, set membership which is the result of an action, e.g. the fact that Capone became a thief, can be represented by treating set membership as an event:

```

{(EV 1002, REL "type", TYPE "membership"),
 (EV 1002, REL "subject", ENT "capone"),
 (EV 1002, REL "object", ENT "thief")}

```

Now we can add the fact that he became a thief in 1908:

```

(EV 1002, REL "year", ENTNUM 1908)

```

Finally, consider set membership which is a consequence of an intrinsic property of an entity, e.g. the triples representing the fact that “Capone stole a car in 1918 in Manhattan” are:

```

{(EV 1004, REL "type", TYPE "steal_ev"),
 (EV 1004, REL "subject", ENT "capone"),
 (EV 1004, REL "object", ENT "car1"),
 (EV 1004, REL "year", ENTNUM 1908),
 (EV 1004, REL "location", ENT "Manhattan")}

```

In the above, we have not represented the fact that car1 is a car. To be consistent, this fact should be represented in a way that is similar to the way in which event 1002 represents the fact that Capone was a thief:

```

{(EV 1006, REL "type", TYPE "membership"),
 (EV 1006, REL "subject", ENT "car1"),
 (EV 1006, REL "object", ENT "car")}

```

It is somewhat burdensome to have to treat membership of a set (e.g. car) which results from the “core” essence of an entity (e.g. car1) in a similar way to membership of a set which is contingent on an action. However, this allows us define denotation of all common nouns in the same way.

From the examples given above, one can see that when set membership (e.g. the set thief) is contingent on an action (e.g. steal), there could be some redundancy in the triple store. For example, the data represented by event 1002 could be derived from event 1004 data. We do not address this concern in this paper, as it has to do with how data from other data structures is converted to triple store data, and what deductive machinery accompanies the triple store.

## 4. RETRIEVING DATA FROM AN EVENT-BASED TRIPLE STORE

Before we introduce the new event-based semantics, we define some basic triple store retrieval operators. Given the limitations of space, rather than define the retrieval operators and our new semantics using the notation of lambda calculus and set theory, and then show how they can be implemented in a programming language, we give the definitions directly using the notation of the programming language Miranda. We choose Miranda for four reasons: 1) It has built-in list operators and a list comprehension construct which corresponds to the “relative set notation” that we used in denotations in FLMS (section 2.3). 2) Similar to MS and FLMS, our new semantics uses higher-order functions which can be defined directly in Miranda. 3) Miranda has a simpler syntax than other higher-order functional languages. 4) Given the declarative nature of Miranda, the definitions are executable specifications which allow us to test our ideas.

In Miranda:

- [x1,..,xn] is a list of n elements of the same type.
- #s is the length of the list s.
- f a1..an returns the result of applying f to a1..an
- member s x returns True if x is in the list s.
- (x1,..,xn) is a tuple of n values of different type.
- Lists are created using list-comprehensions which have the general form: [values|generators;conditions]  
For example: [(x<sup>2</sup> | x <- [1..10], odd x)  
=> [1, 9, 25, 49, 81]
- f a1..an = e defines f to be a function of n arguments whose value is the expression e.
- n \$f m allows f to be used as an infix operator.
- Functions can be composed with the . operator:  
(f . g) x = f (g x)
- map f s applies f to every member of the list s.
- New types can be defined using type constructors, e.g.

```
field ::= EV num      | ENT [char] | ENTNUM num
        TYPE [char] | REL [char] | ANY
```

then EV 1000 is a value of type field

Note in function application brackets are used to establish the order of application, not to enclose arguments, e.g. sqrt 9 + sqrt (2 + 2) => 5. We begin by defining a triple store called data which we use as an example throughout the rest of the paper. Note that we have used type constructors EV, REL etc. (which we earlier referred to as “tags”) in the definition of the triple store. Note also that the definition of data is part of the Miranda program that we built to test our semantics.

```
data =
[(EV 1000, REL "type",      TYPE "born_ev"),
 (EV 1000, REL "subject",   ENT "capone"),
 (EV 1000, REL "year",      ENTNUM 1899),
 (EV 1000, REL "location",  ENT "brooklyn"),
 (EV 1001, REL "type",      TYPE "join_ev"),
 (EV 1001, REL "subject",   ENT "capone"),
```

```
(EV 1001, REL "object",    ENT "fpg"),
 (EV 1002, REL "type",      TYPE "membership"),
 (EV 1002, REL "subject",   ENT "capone"),
 (EV 1002, REL "object",    ENT "thief"),
 (EV 1002, REL "year",      ENTNUM 1908 ),
 (EV 1003, REL "type",      TYPE "smoke_ev"),
 (EV 1003, REL "subject",   ENT "capone"),
 (EV 1003, REL "object",    ENT "bowery"),
 (EV 1004, REL "type",      TYPE "steal_ev"),
 (EV 1004, REL "subject",   ENT "capone"),
 (EV 1004, REL "object",    ENT "car_1"),
 (EV 1004, REL "year",      ENTNUM 1918),
 (EV 1004, REL "location",  ENT "manhattan"),
 (EV 1005, REL "type",      TYPE "smoke_ev"),
 (EV 1005, REL "subject",   ENT "capone"),
 (EV 1006, REL "type",      TYPE "membership"),
 (EV 1006, REL "subject",   ENT "car_1"),
 (EV 1006, REL "object",    ENT "car"),
 (EV 1007, REL "type",      TYPE "membership"),
 (EV 1007, REL "subject",   ENT "fpg"),
 (EV 1007, REL "object",    ENT "gang"),
 (EV 1008, REL "type",      TYPE "membership"),
 (EV 1008, REL "subject",   ENT "bowery"),
 (EV 1008, REL "object",    ENT "gang"),
 (EV 1009, REL "type",      TYPE "join_ev"),
 (EV 1009, REL "subject",   ENT "torrio"),
 (EV 1009, REL "object",    ENT "fpg"),
 (EV 1010, REL "type",      TYPE "membership"),
 (EV 1010, REL "subject",   ENT "capone"),
 (EV 1010, REL "object",    ENT "person"),
 (EV 1011, REL "type",      TYPE "membership"),
 (EV 1011, REL "subject",   ENT "torrio"),
 (EV 1011, REL "object",    ENT "person")]
```

We now define a basic retrieval function getts which returns triples from data which match given field value(s):

```
getts (a,ANY,ANY) = [(x,y,z) | (x,y,z) <- data; x = a]
getts (ANY,ANY,c) = [(x,y,z) | (x,y,z) <- data; z = c]
etc.
```

Example uses are:

```
getts (ANY, "subject", "torrio")
=> [(1009, "subject", "torrio"),
 (1011, "subject", "torrio"),
 etc.]
```

```
getts (1009, "type", ANY) => [(1009, "type", join_ev)]
```

Operators to extract one or more fields from a triple include:

```
first      (a,b,c) = a
second     (a,b,c) = b
third      (a,b,c) = c
thirdwithfirst (a,b,c) = (c, a) etc.
```

Operators which return sets of fields from sets of triples can be defined using the functions above and the function map:

```
firsts trips      = map first trips
thirds trips      = map third trips
thirdswithfirsts trips = map thirdwithfirst trips etc.
```

We can now define more complex operators, such as:

```
get_subj_for_event ev
= thirds (getts (ev, REL "subject", ANY))
```

```
get_subjs_for_events evs
= concat (map get_subj_for_event evs)
```

Such that:

```
get_subjs_for_events [EV 1000, EV 1009]
=> [ENT "capone", ENT "torrio"]
```

The function `get_members` returns all entities which are members of a given set:

```
get_members set = get_subjs_for_events events
where
events_for_type_membership
= firsts (getts (ANY,REL "type",TYPE "membership"))
events_for_set_as_object
= firsts (getts (ANY,REL "object", ENT set))
events
= intersect events_for_type_membership
             events_for_set_as_object
```

An example use of this operator is:

```
get_members "person" => [ENT "capone", ENT "torrio"]
```

Another useful operator is one which returns all of the subjects of an event of a given type:

```
get_subjs_of_event_type event_type
= get_subjs_for_events events
where
events
= firsts (getts (ANY, REL "type", TYPE event_type))
```

For example:

```
get_subjs_of_event_type "smoke" => [ENT "capone"]
```

## 5. A NEW SEMANTICS BASED ON TRIPLES AND EVENTS

### 5.1 Denotations of words

We begin with nouns. As in FLMS, the denotation of a noun is the set of entities which are members of the set associated with that noun. The `get_members` function returns that set as a list. Note that in the Miranda program, sets are implemented as lists. We use the term “set” when discussing the semantics and “list” when discussing the implementation of the triple-store operators. Note also, that from now on, instead of representing denotations as, for example: `||person||`, we define denotations as functions with an appropriate name, e.g. `person`. These denotations can then be applied to each other in the program, as shown on the next page, to create the meanings of more complex phrases.

```
person = get_members "person"
gang   = get_members "gang"
car    = get_members "car"
thief  = get_members "thief"
```

```
e.g. gang => [ENT "fpg", ENT "bowery"]
```

Next, we consider intransitive verbs. The denotation of an intransitive verb is the set of entities which are subjects of an event of the type associated with that verb:

```
smoke = get_subjs_of_event_type "smoke_ev"
```

```
e.g. smoke => [ENT "capone"]
```

Intransitive use of transitive verbs are similar:

```
steal_intrans = get_subjs_of_event_type "steal_ev"
steal_intrans => [ENT "capone"]
```

As in FLMS, proper nouns denote functions which take a set of entities as argument and which return `True` if a particular entity is a member of that set, and `False` otherwise:

```
capone setofents = member setofents (ENT "capone")
torrio setofents = member setofents (ENT "torrio")
car_1 setofents = member setofents (ENT "car_1")
fpg setofents = member setofents (ENT "fpg")
year_1908 setofents = member setofents (ENTNUM 1908)
etc.
An example application: capone smoke => True
```

The quantifiers, “a”, “one”, “two”, “every”, etc. and the conjunctions are defined in the same way as in FLMS:

```
a nph vbph = #(intersect nph vbph) ~ = 0
one nph vbph = #(intersect nph vbph) = 1
two nph vbph = #(intersect nph vbph) = 2
every nph vbph = subset nph vbph
```

```
nounand s t = intersect s t
nounor s t = mkset (s ++ t)
that = nounor
```

```
termand tmph1 tmph2
= f where
f setofevs = (tmph1 setofevs) & (tmph2 setofevs)
termor tmph1 tmph2
= f where
f setofevs = (tmph1 setofevs) \ (tmph2 setofevs)
```

An example application of the above is:

```
(capone $termor torrio) thief => True
```

Transitive verbs are more complex. We need something similar to the `image` in the FLMS approach. We can create “images” for an event `et` using the following:

```
make_image et
= collect
(concat [(thirdswithfirsts . getts)
        (ev, REL "subject", ANY) | ev <- events])
where
events = (firsts . getts) (ANY, REL "type", TYPE et)
```

An example application:

```
make_image "join_ev"
=> [(ENT "capone", [EV 1001, EV 1003]),
    (ENT "torrio", [EV 1009])]
```

We can now use `make_image` to define the denotation of a transitive verb associated with an event of a given type:

```
join
= f where
f tmph
= [subj | (subj, evs) <- make_image "join_ev";
    tmph(concat[(thirds.getts)
                (ev, REL "object", ANY) | ev <- evs])]
```

This definition is somewhat complex. We begin by noting that a termphrase is a syntactic category that includes proper nouns and determiner phrases such as “fpg”, “a gang”, “a gang that was joined by torrio” etc. The denotation of “join” is a function `f` such that when `f` is applied to a termphrase `tmph` (which is itself a function) it returns a list



of subjects each of which is associated with a set of events `evs` in the image of the `join_ev`, such that when `tmph` is applied to the list of objects of the events `evs`, the result is `True`, e.g.:

```
join (a gang) => [ENT "capone", ENT "torrio"]
```

ENT `capone` is in the result owing to the fact that the denotation of the termphrase `(a gang)` is a function which returns `True` when applied to the list of objects of the set of events associated with ENT `"capone"` in the image of event type `join_ev`. Similarly for ENT `"torrio"`.

We can define the passive form of transitive verbs by replacing `subject` by `object` in the definition of `make_image` and use it to create a function `make_passive_trans`. For example:

```
joined_by = make_passive_trans "join_ev"
```

Example use:

```
joined_by (capone $termord torrio) => [ENT "fpg"]
```

We conclude this sub-section by defining some “query” words:

```
which   nph vph   = intersect nph vph
how_many nph vph = intersect nph vph
did      tph vbph = "yes", if tph vbph = True
              = "no", otherwise
who      vph      = which person vph
```

## 5.2 Prepositional phrases

Complex prepositional phrases, such as “in 1908 or 1918 in a city in Illinois” have typically been somewhat difficult to integrate into a compositional NL query semantics which allows arbitrarily-nested quantification (which our semantics does). We do not have any problems and can easily accommodate multiple prepositional phrases by having the parser convert the list of prepositional phrases to a possibly empty list of “prepositional pairs”. Each pair consists of a REL value and a termphrase. For example, the phrase “in 1908 or 1918, in Manhattan” which consists of two prepositional phrases is converted to:

```
[(REL "year", year_1908 $termord year_1918),
 (REL "location", "manhattan")]
```

The definition of each transitive verb is redefined to make use of this list to filter the events which are in the image of the event-type associated with that transitive verb before the termphrase which is the argument to the denotation of the transitive verb is applied to the set of objects associated with the event. A recursive function called `filter_ev` applies each prepositional phrase in turn as a filter to each event:

```
steal' tmph preps
= [ subj | (subj, evs) <- image_steal;
  tmph (concat
    [(thirds.getts) (ev, REL "object", ANY)
     | ev <- evs; filter_ev ev preps])]

filter_ev event [] = True
filter_ev event (prep:list_of_preps)
= ((snd (prep)) ((thirds.getts)
  (event, fst (prep), ANY)))
  & filter_ev event list_of_preps
```

for example:

```
steal' (a car)
[(REL "year", year_1908 $termord year_1918),
 (REL "location", "manhattan")]
```

```
=> [ENT "capone"]
```

## 6. DEFINING WORDS INDIRECTLY AND EXAMPLE QUERIES

The meaning of some words can be defined in terms of words and phrases whose meanings are known. For example:

```
gangster = join (a gang)
```

Our EV-FLMS semantics has the six properties mentioned earlier. The answers to complex queries can be obtained from the meanings of their components by simple function application. For example, the query “Which gangster who stole a car in 1908 or 1918 in Manhattan, joined a gang which was joined by Torrio?” would be converted to the following functional expression by the parser, and then evaluated directly by the programming language in the same way as the expression  $3 + (2 * 4)$  would be evaluated:

```
which
  (person $that
   (steal' (a car)
    [(REL "year", year_1908 $termord year_1918),
     (REL "location", "manhattan")]))
  (join (a (gang $that (joined_by torrio))))

=> [ENT"capone"]
```

The conversion, by the parser, of the word “in” to (REL `"year"`) and (REL `"location"`) in the two different contexts is clumsy and contravenes Montague’s notion that words do not denote entities directly. We will improve this approach in future work.

In our semantics, queries can contain arbitrarily-nested quantification. Termphrases with quantifiers (“a”, “every”, “some”, “one” “two”, etc.) can also appear in prepositional phrases. For example, if the data store held the appropriate triples, the following query can be processed “Who broke into a bar using a jimmy or a brick in two cities located in Illinois?”

## 7. USING A PARSER TO DISAMBIGUATE

Our new semantics has a one-to-one correspondence between the syntax rules defining the syntactic structure of the queries, and the semantic rules determining the order of application of the functional denotations. All phrases and words of a syntactic category have denotations (meanings) of the same semantic type, simplifying integration of the semantics with a parser to create a syntax-directed interpreter. We have already done this for the FLMS semantics and we are currently doing this for the semantics presented here.

There is insufficient space in this paper to discuss ambiguity in detail. In summary, our parser generates more than one syntax tree for ambiguous queries. For example, the query “Did Capone and Torrio join a gang?” would be parsed in two ways, resulting in the two expressions:

```
(capone $termord torrio) (join (a gang))
(a gang) (joined_by (capone $termord torrio))
```

The first returns `True` if Capone and Torrio both joined a, not necessarily the same, gang, and the latter would only return `True` if at least one gang was joined by both Capone and Torrio.

## 8. RELATED WORK

Triple stores have been used in binary-relational databases since the 70's. A comprehensive survey of research on binary relational databases and triplestores, up to and including that carried out in the early 1980's, is provided in [10]. That paper also includes a description of a triple-based query language called WAROUT which appears to be one of the first SPARQL like query languages to have been developed.

Since the 80's, various attempts have been made to create user-friendly query interfaces to binary-relational triplestores. Early attempts include WAROUT mentioned above, pseudo natural-language interfaces [26], Prolog interfaces [29], and graphical visual interfaces [28], [24] and [25].

More recent interfaces to triplestores include the system of Mandreoli et al [23] on flexible query answering which returns best approximations to a query; the four systems (Semantic Crystal, Ginseng, NLP-reduce and Querix) of Kaufmann and Bernstein [1], [20], [19]; the AquaLog system of Lopez et al. [21]; the ORAKEL system of Cimiano et al. [3] which also uses a Montague-like semantics; the NQ system of Ran and Lencevicius [27]; the Pythia system of Unger and Cimiano which converts the NL query to an FLogic query [30]; the SQUALL system of Ferre [8] which is also based on Montague's linguistic approach; the system of Yahya et al [32]; the system of Damova et al. [5] which is also based on a formal logic and which converts NL queries to SPARQL using the Grammatical Framework (GF); the system of Hakimov et al [17] and the Metafrastes system of Embregts et al. [7].

The approach that we have presented in this paper is different from the work mentioned above in that we regard bracketed NL (e.g. English) queries as functional expressions using a formal denotational semantics, and then evaluate those expressions through direct reference to the triplestore using basic triple retrieval operators. We do not translate the NL query to any intermediate language such as SPARQL or FLogic.

This paper describes work which is part of a research project that has extended over several years [9], [13], [15], [11], [16] and [12]. The major contributions of this paper include 1) a detailed explanation of the development of the new semantics, 2) the method for dealing with multiple and complex prepositional phrases, and 3) Miranda program code showing how the event-based semantics can be implemented.

## 9. CONCLUSION AND FUTURE WORK

We have argued that 1) using events rather than entities as the subject of triples, and 2) treating (bracketed) NL queries as expressions of the lambda calculus that can be evaluated directly with respect to the triplestore, allows the creation of a denotational semantics for a wide range of NL queries, and also the construction of query processors as modular syntax-directed interpreters.

The semantics described in this paper is only a proof of concept and much remains to be done.

We have already started work on interfacing our semantics

to remote semantic-web event-based triplestores and have built an on-line query interface. That work is described in an unpublished paper [14].

Our research group is planning to do the following over the next year: 1) improve our approach to prepositional phrases, 2) extend the semantics to accommodate aggregation and negation, 3) integrate the semantics with a parser using the SAIGA attribute grammar programming environment [16], 4) investigate the use of our query processor with existing (conventional) entity-based triple stores in the semantic web. This will require converting, as needed, some of the entity-based triples to event-based triples, 5) investigate the integration of the method of Walter et al [31] for mapping query words to appropriate URIs and building the denotations of words in real-time when the query is parsed, and 6) create a denotational semantics for Japanese and investigate the use of event-based triple stores as an intermediate knowledge representation format for language translation between English and Japanese.

We hope that this paper prompts discussion of the relative advantages and disadvantages of "entity-based" and "event-based" triplestores, and also prompts discussion of the pros and cons of converting NL queries to SPARQL before they are evaluated.

A possible way forward might be to have a 2-stage approach to the development of a powerful NL query processor: Stage I: use the approach described in this paper to develop a formal denotational semantics for a wide range of NL constructs including nested quantifiers, complex chained prepositional phrases, aggregation, negation, modality (such as "who believes that ...", aggregates, and temporal phrases (such as for what period of time...)) Stage II: after the NL semantics has been developed, a translator could be built, based the semantics, to convert NL queries to SPARQL queries. Stage I would facilitate the development of the complex denotational semantics necessary to accommodate a wide range of NL queries, and STAGE II would allow the query processor to make use of the many methods that have been developed to optimize SPARQL queries.

## 10. ACKNOWLEDGMENTS

The authors acknowledge the support of the Natural Science and Engineering Council (NSERC) of Canada, and the reviewers for their comprehensive review and comments on this paper.

## 11. REFERENCES

- [1] A. Bernstein, E. Kaufmann, and C. Kaiser. Querying the semantic web with ginseng: A guided input natural language search engine. In *Proceedings of the 15th Workshop on Information Technology and Systems (WITS 2005)*, pages 45–50, 2005.
- [2] P. Blackburn and J. Bos. *Representation and Inference in Natural Language*. CSLI Publications, 2005.
- [3] P. Cimiano, P. Haase, and J. Heizmann. Porting natural language interfaces between domains: an experimental user study with the orakel system. In *Proceedings of the 12th international conference on Intelligent user interfaces*, pages 180–189. ACM, 2007.
- [4] J. Clifford, S. Abramsky, and C. van Rijsbergen. *Formal Semantics and Pragmatics for Natural Language Querying*. Cambridge Tracts in Theoretical

- Computer Science 8*. Cambridge University Press, Cambridge, 1990.
- [5] M. Damova, D. Dannelles, R. Enache, M. Mateva, and A. Ranta. Natural language interaction with semantic web knowledge bases and lod. In *Towards the Multilingual Semantic Web*. Springer, 2013.
- [6] D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, Yokyo, 1981.
- [7] H. Embregts, V. Milea, and F. Frasincar. Metafrastes: A news ontology-based information querying using natural language processing. In *The 8th International Conference on Knowledge Management in Organizations*, pages 313–324. Springer, 2014.
- [8] S. Ferre. Squall: A controlled natural language for querying and updating rdf graphs. In *Proceedings of CNL 2012*, pages 11–25. LNCS 7427, 2012.
- [9] R. Frost and J. Launchbury. Constructing natural language processors in a lazy functional language. *The Computer Journal*, 32(2):108–121, 1989.
- [10] R. A. Frost. Binary-relational storage structures. *The Computer Journal*, 25(3):358–367, 1982.
- [11] R. A. Frost. Realization of natural language interfaces using lazy functional programming. *ACM Comput. Surv.*, 38(4):1–54, 2006.
- [12] R. A. Frost, B. S. Amour, and R. Fortier. An event based denotational semantics for natural language queries to data represented in triple stores. In *Proceedings of ICSC 2013*. IEEE, Sept. 2013.
- [13] R. A. Frost and P. Boulos. An efficient compositional semantics for natural-language database queries with arbitrarily-nested quantification and negation. In *Conference Proceedings of Advances in Artificial Intelligence, the 15th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2002*, pages 252–267. LNCS 2338, 2002.
- [14] R. A. Frost, J. Donais, E. Matthews, and R. Stewart. A denotational semantics for natural language query interfaces to semantic web triplestores. In *Submitted for publication*, 2014.
- [15] R. A. Frost and R. Fortier. An efficient denotational semantics for natural language database queries. In *Proceedings of Natural Language Processing and Information Systems, 12th International Conference of Applications of Natural Language to Information Systems, NLDB 2007*, pages 12–24. LNCS 4592, 2007.
- [16] R. Hafiz and R. Frost. Lazy combinators for executable specifications of general attribute grammars. In *Proceedings of the 12th International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 167–182. ACM-SIGPLAN, Jan. 2010.
- [17] S. Hakimov, H. Tunc, M. Akimaliev, and E. Dogdu. Semantic question answering system over linked data using relational patterns. In *Proc. of the Joint EDBT/ICDT 2013 Workshops*, pages 83–88. ACM, 2013.
- [18] H. Hendricks. *Studied Flexibility: categories and types in syntax and semantics*. Doctoral Dissertation, Universiteit van Amsterdam, 1993.
- [19] E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics - Science, Services and Agents on the World Wide Web*, 8(4):377–393, Nov 2009.
- [20] E. Kaufmann, A. Bernstein, and R. Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *Proceedings of the 5th International Semantic Web Conference*, Nov 2006.
- [21] V. Lopez, V. Uren, E. Motta, and M. Pasin. Aqualog: An ontology-driven question answering system for organizational semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):72–105, 2007.
- [22] M. G. Main and D. Benson. Denotational semantics for a natural language question answering program. *Computational Linguistics*, 9(1):11–21, 1983.
- [23] F. Mandreoli, R. Martoglia, W. Penzo, and G. Villani. Flexible query answering on graph-modeled data. In *Proceedings of the 12th International Conference on Extending Database Technology*, pages 216–227. EDBT, March 2009.
- [24] J. A. Mariani and R. Lougher. Triplespace an experiment in 3d graphical interface to a binary-relational database. *Interacting with Computers*, 4:147–162, 1992.
- [25] N. Memon and H. Larson. Investigative data mining toolkit: a software prototype for visualizing, analyzing and destabilizing terrorist networks. In *Proceedings Visualizing Network Information, RTO-MP-IST*, pages 1–24, 2006.
- [26] N. Nicholson. *The design of a user-interface to a deductive database: a sentence based approach*. PhD thesis Dept. of Computer Science - Birkbeck College, University of London, 1988.
- [27] A. Ran and R. Lencevicius. Natural language query system for rdf repositories. In *Proceedings of the 7th International Symposium on Natural Language processing*, pages 1–6. SNLP, 2007.
- [28] Smith and King. Incrementally visualizing criminal networks. In *Proceedings of the Sixth International Conference on Information Visualisation*, 2002.
- [29] S. Todd. An interface from prolog to a binary relational database. *Prolog and databases - implementations and new directions*, pages 108–117, 1989.
- [30] C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In *NLDB 2011, LNCS 6716*, pages 153–160, 2011.
- [31] S. Walter, C. Unger, P. Cimiano, and D. Bär. Evaluation of a layered approach to question answering over linked data. In *The Semantic Web-ISWC 2012*, pages 362–374. Springer, 2012.
- [32] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, , and G. Weikum. Natural language questions for the web of data. In *The 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390. ACL, July 2012.

# GraphMCS: Discover the Unknown in Large Data Graphs

Elena Vasilyeva<sup>1</sup>

Maik Thiele<sup>2</sup>

Christof Bornhövd<sup>3</sup>

Wolfgang Lehner<sup>2</sup>

<sup>1</sup>SAP AG  
Dresden, Germany

elena.vasilyeva@sap.com

<sup>2</sup>Database Technology Group  
Technische Universität Dresden, Germany

firstname.lastname@tu-dresden.de

<sup>3</sup>SAP Labs, LLC  
Palo Alto, USA

christof.bornhoevd@sap.com

## ABSTRACT

Graph databases implementing the property graph model provide schema-flexible storage and support complex, expressive queries like shortest path, reachability, and graph isomorphism queries. However, both the flexibility and expressiveness in these queries come with additional costs: queries can result in an unexpected, empty answer. To understand the reason of an empty answer, a user normally has to create alternative queries, which is a cumbersome and time-consuming task.

To address this, we introduce diff-queries, a new kind of graph queries, that give an answer about which part of a query graph is represented in a data graph and which part is missing. We propose a new algorithm for processing diff-queries, which detects maximum common subgraphs between a query graph and a data graph and computes the difference between them. In addition, we present several extensions and optimizations for an established maximum common subgraph algorithm for processing property graphs, which are the foundation of state of the art graph databases.

## Categories and Subject Descriptors

H.2.4 [Systems]: Query processing

## Keywords

Maximum Common Subgraph, Flexible Query Answering

## 1. INTRODUCTION

New kinds of data and their analysis increase the demand for flexible data models supporting data of different degrees of a structure. Graph databases implementing the property graph model [11] are a reasonable answer to this demand. They support diverse data with different degrees of a structure in the form of a graph. A diverse schema of vertices and edges is represented by an arbitrary number of attributes,

which can differ between vertices or edges of the same semantic type. A major advantage is that such systems do not require a predefined rigid database schema.

However, the flexibility provided by graph databases and the property graph model comes with additional costs. A user of graph databases typically has only limited knowledge about stored data, which complicates the creation of queries. He can overspecify queries that can result in an empty answer. Any empty response causes confusion on the user side, since its reason is unclear: was the query overspecified or are some data missing in the database? To answer this question, a user needs a possibility for explorative queries and guidance through the query answering process. To provide this, a system has to be able to give intermediate points in query processing, which describe the already discovered and still missing parts of a query graph. As a result, a user could discover overspecified query parts or conclude that some information is missing in a dataset and, therefore, has to be obtained from external data sources [8, 15].

### *Related Work.*

If the result of a query does not meet the user's expectations, he can conduct "Why Not?" queries [3] to determine why the result set does not include the items of interest. It is assumed that a user cannot process the data manually because of their large volume and complexity. A user specifies items of interest with attributes or key values. Then a "Why Not?" query could be "Why are the items with predicate P not in the result set?"

There are several ways of answering "Why Not?" queries. On the one hand, the causes for an empty answer can be found, as done in [3], where a "Why Not?" query applies a set of manipulations to the original query. As an answer, the system provides an operator from the original query, which removes required items from the result set. This approach relies on manipulations of operators and derives an answer for a specific item. On the other hand, a provenance-based explanation can be delivered by computing the provenance of possible answers for SPJ queries, like for example in [9]. This is also possible to refine the query in such a way, that the items of interest appear in the result set. In this case, the explanation for a "Why Not?" query is based on an automatically generated query, which response consists of the original results and the items of interest [13].

In contrast, our problem is to find missing structural parts of a query that prevent the system from delivering a non-empty answer. At this point, we are not interested in specific attributes and items (which is done in the relational case).

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

To the best of our knowledge, the question of discovering a missing query part in a data graph has not been addressed in graph database research.

### Contributions.

In graph databases, a query can be understood as a pattern that has to be sought in a large data graph. To tell a user which query parts were discovered in a data graph and which are missing, we propose (1) to find maximum common subgraphs in a data graph for a given query, and (2) to calculate the difference between them and a query graph. As a result, the system yields a list of discovered maximum common subgraphs as starting points and undiscovered parts of a query graph as a subject for the future explorative search.

As our contributions we present in this paper diff-queries, a new kind of graph database queries, which give an answer about existing and missing query parts. Diff-queries deliver discovered maximum common subgraphs of a data graph and corresponding missing parts of a query graph. We introduce an all-covering spanning tree allowing for the processing of a whole query graph and weakly connected graphs. This tree allows us to get larger subgraphs than the standard solutions for connected graphs. We also provide several optimizations for diff-queries to deliver a final result faster and to reduce the number of intermediate subgraphs.

The rest of the paper is structured as follows. We introduce the property graph model and basic algorithms for discovering maximum common connected subgraphs between two graphs in Section 2. We outline the processing of diff-queries in a graph database in Section 3. In Section 4 we describe challenges of multiple starts and weakly connected graphs for the standard algorithm and their solutions. We provide several optimizations for our proposed algorithm in Section 5 and evaluate our approach in Section 6.

## 2. MAXIMUM COMMON CONNECTED SUBGRAPH DETECTION

As an underlying data model we use the property graph model [11]. It represents a graph as a directed multigraph, where vertices are entities and edges are relationships between them. Each edge and vertex can be described by multiple attributes and their values. The attributes can differ concerning edges and vertices – even if they are of the same semantic type. We define a **property graph** as a directed graph  $G = (V, E, u, f, g)$  over attribute space  $A = A_V \dot{\cup} A_E$ , where: (1)  $V, E$  are finite sets of vertices and edges; (2)  $u : E \rightarrow V^2$  is a mapping between edges and vertices; (3)  $f(V)$  and  $g(E)$  are attribute functions for vertices and edges; and (4)  $A_V$  and  $A_E$  are their attribute space.

A graph  $G' = (V', E', u', f', g')$  is a **connected subgraph** of  $G$ , if  $V' \subseteq V, E' \subseteq E, u' |_{u}, f' |_{f}$ , and  $g' |_{g}$ .

Given a data graph  $G_d$  and a query graph  $G_q$ , the graph  $G'_d = (V'_d, E'_d, u'_d, f'_d, g'_d)$  is a **common connected subgraph** of graphs  $G_d$  and  $G_q$ , if  $G'_d$  is a connected subgraph of  $G_d$  and  $G'_d$  is a connected subgraph of  $G_q$ . There may be multiple common connected subgraphs in a data graph  $G_d$  for a query graph  $G_q$ .

For property graphs, a **maximum common connected subgraph**  $G'_d$  is a common connected subgraph of a data graph  $G_d$  for a query graph  $G_q$  such that there exists a match  $S_{max}$  in  $G_d$  for  $G_q$  such that for any match  $S$  in  $G_d$  for  $G_q$ ,  $S \leq S_{max} : V \leq V_{max} \cup E \leq E_{max}$ .

### Finding Maximum Common Connected Subgraphs.

To tell, which part of a query can be found in a data graph, we have to find maximum common subgraphs in a data graph  $G_d$  for a query graph  $G_q$ . This can be done by maximum common connected subgraph algorithms. The computation depends on how a graph is stored and processed. A commonly used adjacency matrix or adjacency list allow the compact storing of graphs and their efficient processing [5]. For example, a matrix  $M$  consists of  $n \times n$  elements, where  $n$  is the number of vertices in a graph. Each element of a matrix  $a_{ij}$  with a value 1 represents an edge between vertices  $i$  and  $j$ . A maximum common connected subgraph can be calculated by linear algebra operations. If a graph is a property graph, then its attributes can be stored in separated structures and can be used during prefiltering.

Ullmann in [14] describes a brute-force tree-search enumeration procedure, which efficiently eliminates successor vertices. It excludes some elements from a matrix  $M$  and, thereby, reduces the search space. The algorithm is commonly used for exact graph matching. Another backtracking algorithm – the McGregor algorithm [10] – also works on matrices and provides extension points for pruning techniques and prefiltering options.

Both methods rely on labeled graphs, which differ from our underlying property graph model [11]. To apply them to our use case, these algorithms have to be adapted to work on properties on edges and vertices.

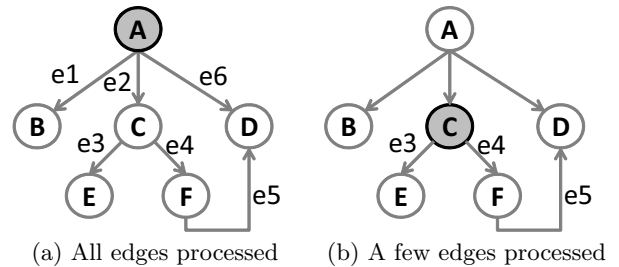


Figure 1: Depth-first search

Ullmann’s [14] and McGregor’s [10] algorithms are backtracking algorithms, which are a base for traversal operations in graph databases. Both methods implement a depth-first search that begins at the root and traverses the graph as far as possible along each branch before backtracking. Assuming the search starts from the grey vertex, in the example shown in Figure 1(a) we begin then from vertex  $A$  and explore all edges of the graph as follows:  $e1, e2, e3, e4, e5, e6$ . If we start from vertex  $C$  like in Figure 1(b), then only edges  $e3, e4, e5$  are traversed. To ensure the discovery of all maximum common connected subgraphs, the search is conducted for each vertex of a query, and a data graph is treated as undirected. This makes the search NP-complete.

A maximum common connected subgraph problem can also be modified for the search of a maximum clique like in the Durand-Pasari algorithm [7] and in the Balas Yu algorithm [1]. These algorithms are also tree-search algorithms. Some of them work better with sparse graphs, others with dense graphs. According to [4], the McGregor algorithm shows good results in all cases and has the best space complexity. Based on these observations, we have chosen it as the base for our discovery of maximum common connected subgraphs.

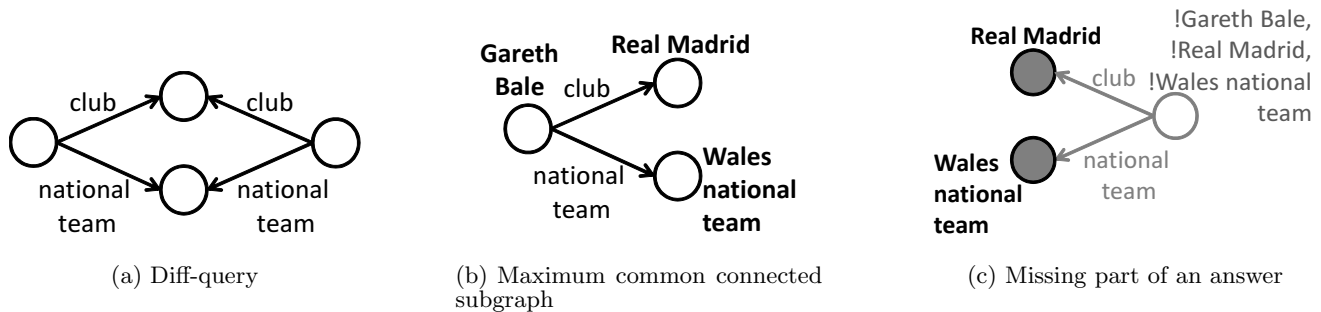


Figure 2: A diff-query and its answer: which two vertices play in the same national team and club?

### Difference Graphs.

With a maximum common connected subgraph algorithm we can determine, which query part has an answer in a data graph. To detect, which structural part is missing we need to compute the difference graph – the difference between discovered maximum connected subgraphs and a query graph.

A difference graph includes those query vertices and edges, which were not discovered during query processing, and the instances of query vertices adjacent to a maximum common connected subgraph.

For property graphs we define a **difference graph** as a graph  $G'_q = (V'_q, E'_q, u'_q, f'_q, g'_q, V'_d(adj), C)$ , where  $V'_q \subseteq V_q, E'_q \subseteq E_q, u'_q |_{u_q}, f'_q |_{f_q}, g'_q |_{g_q}, V'_d(adj)$  are adjacent vertices, and  $C$  is a set of non-adjacent discovered vertices to be excluded from the further explorative search.

In [6] the authors compute difference graphs from conceptual graphs. The first graph is transformed into one of its large common subgraphs and the set of applied operations is stored. Then the large common subgraph is transformed into the second graph, and the set of applied operations is stored. Finally, both stored sets are concatenated in a difference graph. Operations include standard insert and remove operations and specific operations for conceptual modeling like a generalization or a specialization of a concept.

In our work we do not provide any conceptual analysis and reasoning for “non-existing” edges from a hierarchical taxonomy. Moreover, the first step is redundant: we record the discovered parts of a graph during the graph traversal.

### Diff-queries.

If a user gets an empty response to his query, he can conduct a **diff-query** that shows, which query part is addressed in data and which part is missing. For this purpose, it detects maximum common connected subgraphs and computes their corresponding difference graphs, which prevent a system from the delivery of a non-empty answer to a user.

Assuming we search for two soccer players from the same national team and the same club. Then the query graph could be represented as in Figure 2(a). A possible answer to this diff-query would consist of a maximum common connected subgraph  $G'_d$  as shown in Figure 2(b), and a missing part of a query with constraints  $G'_q$  as in Figure 2(c). The first part includes all discovered instances of vertices and edges like “Gareth Bale”, “Real Madrid”, and “Wales national team”. The second part consists of instances of discovered adjacent vertices (dark grey), missing query vertices and edges (grey), and constraints for vertices (grey).

## 3. PROCESSING OF DIFF-QUERIES IN GRAPH DATABASES

In this section we shortly describe the graph database we use in our prototype and outline the diff-query computation process. The processing of diff-queries consists of two steps: the detection of a maximum common connected subgraph by using an extended version of the McGregor algorithm [10] for property graphs, and the computation of a difference graph between the discovered maximum common connected subgraph and a query graph.

### 3.1 Storage Representation

In our prototype system the property graph model is implemented as a graph abstraction on top of a RDBMS, which uses separate tables for vertices and edges. Vertices are described by a set of columns for their attributes, and edges are stored as simplified adjacency lists in a table. Each edge can have multiple attributes, which are stored together with its description. All edges and vertices have unique identifiers.

To process such a graph efficiently, we use an in-memory column database, which supports optimized flexible tables (new attributes can efficiently be added and removed) and provides advanced compression techniques for sparsely populated columns like in [2, 12]. This abstraction allows us to store graphs with an arbitrary number of attributes without a predefined rigid schema.

The graph database provides the following operations: insert, delete, update, filter based on attribute values, aggregation, and graph traversal in a breadth-first manner. Traversal along directed edges is possible in both directions with the same performance.

Queries to the database are represented via graphs, where vertices describe entities and edges describe connections between them. Each description of vertices and edges can include predicates for attribute values. A specific vertex is represented by its identifier in a query graph.

### 3.2 Detection of Maximum Common Connected Subgraphs

To detect the maximum commonality between a query and a data graph, we have chosen the McGregor maximum common connected subgraph algorithm [10] presented in Algorithm 1, which uses a depth-first search (Figure 1).

To leverage the McGregor algorithm for property graphs, the edges and vertices tables of our graph database have to be processed. First, the projection on a vertices table reduces the number of start vertices at line 4. Second, each

---

**Algorithm 1** The MCCS algorithm for a property graph

---

```
1: function MCCSSEARCH(query graph  $G_q$ )
2:    $graphs, tmp$ 
3:   for all edge  $q_i$  in  $G_q$  do
4:      $sources_i = getSourceVertices(q_i)$ 
5:      $graph = DFS(sources_i, q_i, true, graphs_i)$ 
6:      $graphs.addGraph(graph)$ 
7:   for all  $graphs_i$  do
8:     if  $graphs_i > tmp$  then  $tmp = graphs_i$ 
9:   return  $tmp$ 
10: /*depth-first search*/
11: function DFS( $sources, edge, isStart, graph$ )
12:   for all  $sources_j$  do
13:     if  $isStart$  then  $edge = getNextEdge(edge)$ 
14:     if  $noNextEdge$  then return  $graph$ 
15:      $targets = traverse(edge)$ 
16:      $filterTargets(targets)$ 
17:     for all  $targets_d$  do
18:        $graph.addEdge(edge, sources_j, targets_d)$ 
19:        $graph = DFS(targets_d, edge, false, graph)$ 
20: return  $graph$ 
```

---

step is traversed by the graph traversal operator at line 15. Finally, the target vertices are filtered according to their predicates (see line 16). To ensure that the algorithm finds a maximum common connected subgraph, it is started multiple times from all query vertices as starting points at line 5. The maximum common connected subgraph is stored for each starting point in a set. After all runs the best subgraph is chosen from the collected set (see lines 7-9).

### 3.3 Computation of a Difference Graph

To compute a missing part of a query, we use a query graph and a discovered maximum common connected subgraph. The process consists of two steps: (1) the split of discovered and undiscovered vertices and edges, and (2) the completion of an undiscovered part with attributes or vertices conditions.

In our first step, during processing we store the mapping between data edges and query edges, data vertices and query vertices in temporary tables. The difference graph consists of query edges and vertices, which are not presented in these temporary tables. Some edges in the difference graph will have only single vertices at their ends, because other end vertices have already been traversed. Therefore, we have to include the discovered edges' ends into the difference graph in the second step – the completion of the difference graph with attributes or vertices conditions.

In the second step we detect, which conditions have to be applied to the graph discovered in the first step. We study the table with discovered vertices and a query description and assign conditions to the difference graph according to several rules: If a query edge is not discovered, but at least one of its end vertices has already been found, then this is a positive condition. It means we include a discovered end vertex into the difference graph. In the example presented above the two dark grey vertices represent such conditions (see Figure 2(c)). Such vertices are included into a difference graph and can be used as starting points for a future explorative search. If a query vertex and all its query edges



Figure 3: Weakly connected graphs

(incoming and outgoing) are discovered in a data graph, then this vertex is a negative condition, and its instance has to be excluded from the non-discovered query vertices. In our example this can be “Gareth Bale” (see Figure 2(c)), which does not have to be considered in a future explorative search.

## 4. PROBLEMS OF MULTIPLE STARTS AND WEAKLY CONNECTED GRAPHS

The general version of the McGregor algorithm [10] takes all vertices of a query as starting points and iterates through them. So, the system traverses the same data edges multiple times. On the one hand, this ensures that no edge is left out and all maximum common connected subgraphs are discovered. On the other hand, this generates large intermediate results and increases the response time dramatically.

We figure out two problems, which solution can increase the performance of the algorithm, find larger graphs, and reduce the number of runs. First, the algorithm works only with connected graphs, therefore, only one-directed search for directed graphs is done. This can be solved by the extension of the search for weakly connected graphs. Second, we can miss some maximum common connected subgraphs by start from a single vertex. This can be solved by a restart strategy for non-traversed edges.

### Processing of Weakly Connected Graphs.

The McGregor maximum common connected subgraph algorithm, which is a base for our algorithm, processes the directed graph only in a forward direction. This can limit the size of discovered subgraphs and deliver subgraphs of potentially smaller size than could be determined. To ensure the discovery of a maximum subgraph, we have to choose that vertex as a root, where all vertices can be reached from. Because the algorithm works only in a forward direction, it is not always possible to find the best root vertex.

For example, the query presented in Figure 3 does not have any ideal root. This is a weakly connected graph: it is connected, if directions of edges are not considered. For this query the McGregor algorithm can discover subgraphs only with two edges and three vertices ( $ABC$  or  $BCD$ ). Therefore, we need to modify the algorithm to also consider unreachable components.

To process queries with unreachable components, we introduce an all-covering spanning tree that has the following characteristics. If the whole query graph is available in data, then the all-covering spanning tree is able to cover all query vertices and edges in a single run. An edge can be included into the search in forward or backward directions. In case of a backward direction, an edge is marked with a flag “back”. This can be done without additional effort because of the underlying data model and the graph traversal operator provided by the database like in [12]. Another way would be to make a graph basically undirected with duplicated data or double table scans, which is less efficient.

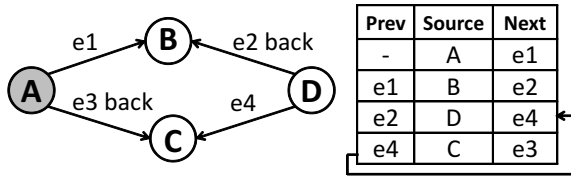


Figure 4: All-covering spanning tree and the backtracking procedure

We adapt Algorithm 1 to work with an all-covering spanning tree. From now on, we consider all edges for each vertex. Outgoing edges have priority over incoming edges and are processed first. After all outgoing edges are traversed, incoming edges are considered.

To guarantee a correct search, we maintain the all-covering spanning tree as a temporary table and refer to it during the backtracking procedure. It records the mapping between previously traversed and next edges. The all-covering spanning tree has three columns: a previous edge, a source vertex, and a next edge. To save space, we can use a Boolean identifier of a traversed edge instead of an identifier for a source vertex. For ensuring the simplicity of explanation, we use vertices in the following example.

Assuming the search for a query presented in Figure 4 begins from vertex  $A$ . At initialization, the spanning tree is empty. Vertex  $A$  has two outgoing edges. After we have followed edge  $e1$ , we add the following entry into the table: no previous edge, source vertex is  $A$ , next edge is  $e1$  ( $-;A;e1$ ). Now we are at vertex  $B$  without any outgoing edges. We take incoming edge  $e2$ , mark it with “back”, and add an entry into the table ( $e1; B; e2$ ). We repeat the process and traverse edges  $e4, e3$ . Finally, we are at vertex  $A$  without any non-traversed edges and start the backtracking.

The backtracking procedure is done according to the created all-covering spanning tree. The last traversed edge is  $e3$ . We check its entry (column “Next”) in the mapping table, take its previous edge  $e4$  and go to source vertex  $C$ . There are no other non-traversed edges for vertex  $C$ , and so we continue the backtracking. The predecessor of  $e4$  is  $e2$  with vertex  $D$ , so we move to it. The procedure continues until it gets to source vertex  $A$ , where no further non-traversed edges exist.

A graph database gives the possibility of changing the direction through suitable storing and processing of edges. All edges are stored in a forward direction – “from a source to a target”. In case of the backward traversal, the graph traversal operator changes the order of columns to be searched: “from a target to a source”. Therefore, we need only to change the direction of an edge in the query description and pass it to the traversal operator.

### Restart Strategy.

With the all-covering spanning tree, we can construct a traversal path, which includes all vertices and edges, and process weakly connected graphs. Hereby, we solve the first problem of the one-directed search. Now we do not need to iterate through all vertices multiple times. We just take one vertex and search from it. This approach works well if all edges are represented in a data graph. The absence of some edges in the data graph can split a query graph into several subgraphs, which are unreachable from each other.

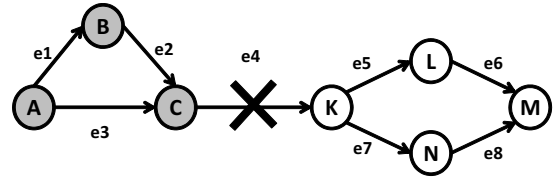


Figure 5: Only white or grey part is traversed

In this case if we start with a vertex from a smaller subgraph, we will miss a maximum common connected subgraph from another subgraph. Thereby, we come to the second problem of the algorithm, which can be solved by a restart strategy.

If we start a search from a single node that is located in the smaller connected subgraph of a query graph, we can potentially miss the larger subgraph, provided by another subgraph. The problem can be explained with a query graph containing a bridge. If a query has a bridge (see Figure 5), which is not addressed in the data graph, then only a subset of vertices and edges is traversed. In our example query edge  $e4$  does not have any matching data edges. In this case, a maximum common connected subgraph found by our algorithm would be the white or the dark-grey part. Therefore, if we do a single run in the dark area the maximum common connected subgraph will be missing, which is located in the white area. To solve this problem, we can resume the search with the edges, which were not traversed. The final maximum common subgraph would be unconnected and would contain all discovered maximum common connected subgraphs.

We maintain a list of traversed edges of a query graph. After the first set of maximum common connected subgraphs is returned, we remove those edges from the list that have already been traversed. The next step is taken from this set. This strategy ensures the discovery of a maximum common subgraph for a given start vertex, if an all-covering spanning tree was constructed.

For example, at the beginning a query graph in Figure 5 has an empty list of traversed edges. Assuming we start from the edge  $e1$  and find edges  $e1, e2, e3$ , but the edge  $e4$  is missing from a data graph. We add all four edges into the list of traversed edges and remove them from the list of start edges. We choose the next start among the edges  $e5, e6, e7, e8$ . The search from any of them will find the same subgraph of four edges. This is the maximum common connected subgraph. If we concatenate it with the first discovered maximum common connected subgraph, then we will get a maximum common unconnected subgraph. So, as a maximum common subgraph we will get a set of unconnected parts. This reduces the number of intermediate results and gives a notion to a user about which edges should exist to complete the graph. Such a methodology can potentially return larger subgraphs than the strategy for connected subgraphs.

Therefore, with all-covering spanning trees and restart strategies we can limit the number of restarts and find a maximum common unconnected subgraph, which then can be used for the further explorative search and the integration of missing data. In the following we use maximum common connected and unconnected subgraphs and refer to them jointly as maximum common subgraphs.

## 5. OPTIMIZATION STRATEGIES



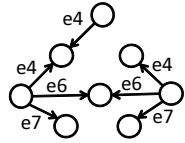
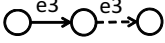
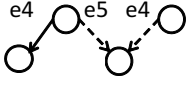
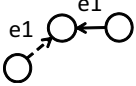
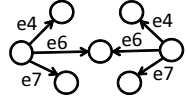
Experiment	Configuration	Topology: Path	Topology: Zigzag	Topology: Star	Optimization
Figure	Figure 6	Figure 7(a)	Figure 7(b)	Figures 7(c)-7(d)	Figures 7(e)-7(f)
Query					
Edge Types	e4, e6, e7	e3	e4, e5	e1	e4, e6, e7

Table 1: Diff-query templates used in the evaluation

To increase the efficiency of the proposed algorithm, we have developed several optimizations for start and restart vertices, and early termination conditions.

### 5.1 Choice of Start and Restart Edges

The general McGregor algorithm [10] processes a graph from all query vertices, and then the biggest graph is chosen and delivered as a maximum common connected subgraph. With all-covering spanning tree and multiple restarts as proposed in Section 4, we can ensure that from each query vertex the whole query can potentially be traversed in the best case. The question is: Which query vertex should be taken as a start? The size of the intermediate results strongly depends on the cardinality of the processed edges and vertices. If a query graph is described very generally, then it will result in a large amount of intermediate results. To decrease it, we extend our algorithm with several strategies to select a start vertex, a start edge, and a next branch to traverse. For example, we can make decisions based on the cardinality of predicates or on the degree of a vertex.

#### Number of Incoming/Outgoing Edges.

The order of edges can be chosen according to the number of their previous or next edges. A vertex with the maximal degree is selected as the starting point. For a vertex with a higher degree, more edges need to be processed, and, therefore, we can discover a maximum common subgraph earlier. This strategy can potentially reduce the number of restarts.

#### Edge and Vertex Cardinality.

Before executing a query, the system calculates the cardinality for all vertices and edges in a query. It then sorts them separately according to the number of items, which should be returned by a system in an ascending order. We choose the edge with the lowest cardinality as the start edge. If we use an all-covering spanning tree, then we can also choose a search direction, based on the cardinality of a source and a target. Otherwise, we use forward processing as default. The same strategy can be applied to restarts, but only the cardinality for edges is considered. In addition, this method has the advantage that if an edge has cardinality = 0 then it is discarded from the search. This reduces the number of table scans and makes the search more efficient.

### 5.2 Threshold-based Termination Condition

In general, the search stops when no more edges are found and a backtracking procedure returns to start. In addition, there can be cases, when a system can stop the search earlier.

A threshold can be an estimated size of a maximum common subgraph or the number of discovered maximum common subgraphs, which could be derived from a data graph. To calculate these numbers, we can reuse the above presented cardinality of a query. If a query graph has  $N$  edges, and for  $M$  edges the  $cardinality(M) > 0$ , where  $M \in N$ , then the maximum common subgraph can have only  $M$  edges. After  $M$  edges are found, the search can be safely stopped. Similar rules can be formulated for sources and targets.

Assuming we have a query with four vertices and three edges with the following predicate cardinalities:  $card_{edge1} = 5$ ,  $card_{edge2} = 2$ ,  $card_{edge3} = 0$ , then the maximum common subgraph can only consist of up to two edges, and we can have a maximum of five graphs like this. We can terminate our search, after the first subgraph with two edges has been discovered.

## 6. EVALUATION

In this section we evaluate diff-queries and proposed optimization techniques. We describe the evaluation setup in Section 6.1. Then we discuss the scalability of the best configuration, derived in Section 6.2, for different query topologies in Section 6.3. Finally, we evaluate start and restart strategies in Section 6.4.

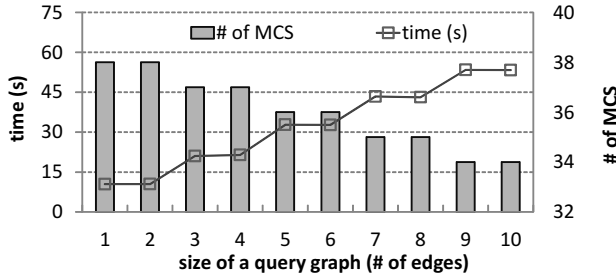
### 6.1 Evaluation Setup

We have implemented our algorithm and its optimizations in an in-memory column database, which provides the graph abstraction as described in Section 3.1. We have created a property graph from DBpedia RDF triples, where labels represent attribute values of entities. It has about 20K vertices and 100K edges. The evaluated queries are presented in Table 1. We have tested each case for each query ten times and have taken the average as a measure.

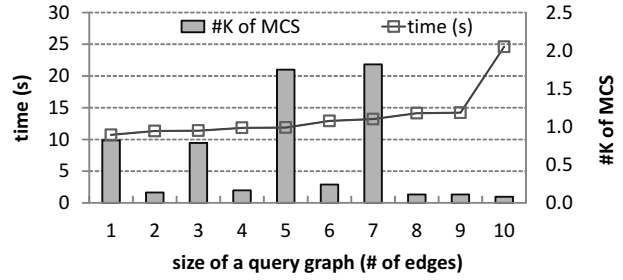
### 6.2 Configuration

In this section we study several configurations of the algorithm: multiple starts from all edges without the all-covering spanning tree (only for connected components), with the all-covering spanning tree (for weakly connected components), and restart strategy (also for unconnected components).

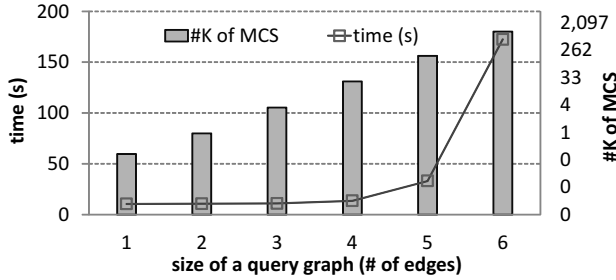
As we can see in Figure 6, the restart strategy discovers larger graphs with shorter response times and less intermediate and final results. Although the method with the all-covering spanning tree has a longer response time (because of the tree construction), it discovers larger graphs. The response time and the size of the maximum common subgraph (MCS) are the best for the restart strategy with the tree construction.



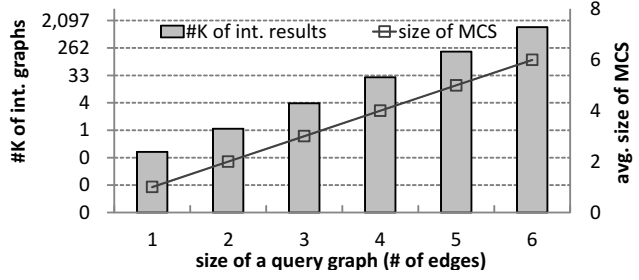
(a) Topology: path



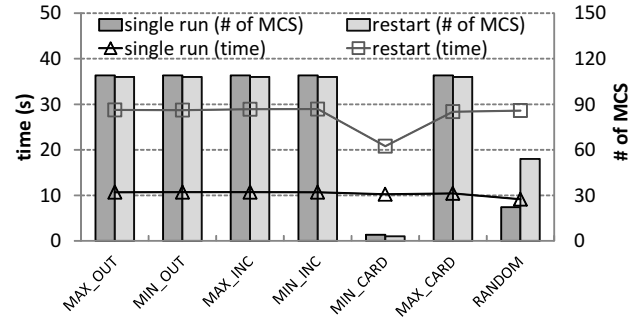
(b) Topology: zigzag



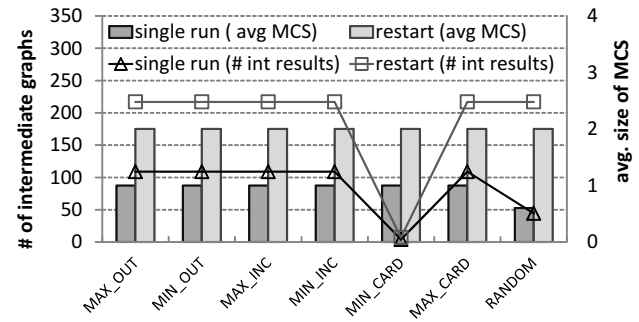
(c) Topology: star



(d) Intermediate results for a star



(e) Optimization: response time evaluation



(f) Optimization: intermediate results

Figure 7: Evaluation of different topologies and optimization strategies

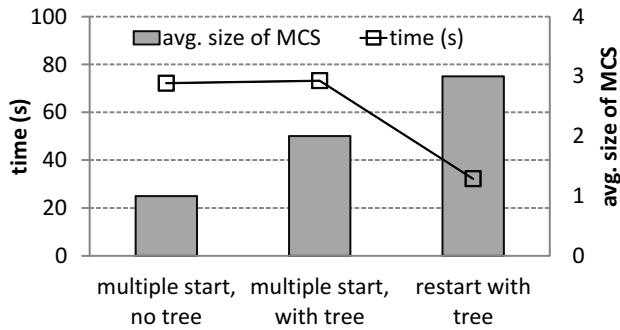


Figure 6: Evaluation of configurations

### 6.3 Topology

In this section we use the best configuration of the previous step: restart with the all-covering spanning tree. For its evaluation on different graph topologies, we have const-

rued several queries, which consist of the edges of similar semantic (for a specific topology). The star and path topologies use a single type of edges, while the zigzag evaluates queries with two edge types. The evaluation results are presented in Figures 7(a)-7(d).

In the path each second edge is missing, so the number of MCS decreases. The star tends to increase the number of solutions, because all edges have the same starting point and larger graphs are combined from smaller graphs. The number of MCS for the zigzag evolves dramatically, because we use edges of two types, otherwise, the behavior would be similar to behavior of the path. With the size of a query graph, the response time is growing linearly, except the star, which is explained by the growing intermediate results (see Figures 7(c)-7(d)).

Comparing the results of the evaluation on three topologies, we conclude that, first, the response time dependency on the number of MCS is linear. Second, for the star topology, the size of a result set grows if edges of the same semantic type are used. Third, the response time depends on both factors: size of intermediate results and size of a query

graph. Fourth, to ensure the linear dependency, optimization strategies have to be used, which reduce the number of intermediate results.

## 6.4 Optimization Strategies

We evaluate start strategies for a single start and for restart (see Figures 7(e)-7(f)). We observe that with the restarts we can increase the average size of MCS. Regardless of the strategy, we get MCS of the same size by using the restart configuration. The response time for the search can be reduced by using an appropriate optimization strategy. For example, the “maximum cardinality” strategy reduces the number of intermediate results, the number of MCS, and the processing time. Although the “random strategy” gives less intermediate results, on average it discovers smaller MCS.

Comparing the results of this evaluation, we conclude that with the restart strategy we can find bigger common subgraphs without starting with each edge multiple times. Optimizations can reduce the number of intermediate results, the number of MCS, and the response time. If characteristics of edges (degree, predicate) are similar, all strategies provide similar results. The strategies of cardinality can be even more efficient, if after the edge selection the direction of its processing is chosen according to the vertices’ cardinality.

With the evaluation we show that the restart configuration and all-covering spanning tree can be used without the start from each query edge. They facilitate to find bigger maximum common unconnected subgraphs with less response time. Optimizations can also decrease the response time, but they will give less MCS.

## 7. CONCLUSION

To express graph queries correctly is a complicated task, because of the diversity and schema flexibility of a data graph. If a query derives an empty answer, a user requires support to understand, what the reason was: an overspecified query or missing data. In this paper we introduce *diff-queries*, a new kind of graph queries, that support a user in such cases. The response to a diff-query describes the parts of a query graph that are addressed and those that are missing in a data graph. The processing of a diff-query consists of two steps: the discovery of a maximum common subgraph and the computation of a difference graph. As a base algorithm we take the McGregor maximum common connected subgraph algorithm. We adapt it for directed weakly-connected property graphs with an all-covering spanning tree and reduce the number of lookups with the restart strategy, which searches from a single edge, does restarts, if some of the edges were not processed, and delivers a maximum common unconnected subgraph. We show that this can be improved by the choice of a start and restart vertex and edge. After the answer is delivered to a user, he can do explorative search of missing data in external sources or modify the query according to the derived difference graph.

Although our method shows good results for our use case, there is an open challenge for the future: the number of intermediate and final results is still very large. We want to develop strategies for reducing and ranking them. In addition, we did not study, how to present and to rate answers according to a given specification. For this purpose, we propose assigning priorities to specific subgraphs of a diff-query and conducting a user study to qualify solutions. Also, we would

like to introduce a similarity measure to quantify vertices, edges and their predicates, and to enhance the algorithm by discarding the backtracking part and by introducing more sophisticated strategies for the choice of a start vertex to decrease the number of restarts.

## 8. ACKNOWLEDGMENT

This work has been supported by the FP7 EU project LinkedDesign (grant agreement no. 284613).

## 9. REFERENCES

- [1] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068, Nov. 1986.
- [2] C. Bornhövd, R. Kubis, W. Lehner, H. Voigt, and H. Werner. Flexible Information Management, Exploration and Analysis in SAP HANA. In *DATA*, pages 15–28, 2012.
- [3] A. Chapman and H. V. Jagadish. Why not? In *Proc. of ACM SIGMOD*, pages 523–534, New York, NY, USA, 2009. ACM.
- [4] D. Conte, P. Foggia, and M. Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *J. Graph Algorithms Appl.*, 11(1):99–143, 2007.
- [5] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.
- [6] H. S. Delugach and A. D. Moor. Difference graphs. In *In Contributions to ICCS 2005*, pages 41–53, 2005.
- [7] P. J. Durand, R. Pasari, J. W. Baker, and C.-c. Tsai. An efficient algorithm for similarity analysis of molecules. *Internet Journal of Chemistry*, 2(17):1–16, 1999.
- [8] J. Eberius, M. Thiele, K. Braunschweig, and W. Lehner. DrillBeyond: enabling business analysts to explore the web of open data. *Proc. VLDB Endow.*, 5(12):1978–1981, 2012.
- [9] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *Proc. VLDB Endow.*, 1(1):736–747, Aug. 2008.
- [10] J. J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12(1):23–34, 1982.
- [11] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Inf. Science and Technology*, 36(6):35–41, 2010.
- [12] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner. The Graph Story of the SAP HANA Database. In *BTW*, pages 403–420, 2013.
- [13] Q. T. Tran and C.-Y. Chan. How to conquer why-not questions. In *Proc of ACM SIGMOD*, pages 15–26, New York, NY, USA, 2010. ACM.
- [14] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, Jan. 1976.
- [15] E. Vasilyeva, M. Thiele, C. Bornhövd, and W. Lehner. Leveraging flexible data management with graph databases. In *GRADES*, pages 12:1–12:6, New York, NY, USA, 2013. ACM.

# Graph-driven Exploration of Relational Databases for Efficient Keyword Search

Roberto De Virgilio  
Dipartimento di Ingegneria  
Università Roma Tre  
Rome, Italy  
dvr@dia.uniroma3.it

Antonio Maccioni  
Dipartimento di Ingegneria  
Università Roma Tre  
Rome, Italy  
maccioni@dia.uniroma3.it

Riccardo Torlone  
Dipartimento di Ingegneria  
Università Roma Tre  
Rome, Italy  
torlone@dia.uniroma3.it

## ABSTRACT

Keyword-based search is becoming the standard way to access any kind of information and it is considered today an important add-on of relational database management systems. The approaches to keyword search over relational data usually rely on a two-step strategy in which, first, tree-shaped answers are built by connecting tuples matching the given keywords and, then, potential answers are ranked according to some relevance criteria. In this paper, we illustrate a novel technique to this problem that aims, rather, at generating directly the best answers. This is done by representing relational data as graph and by combining progressively the shortest join paths that involve the tuples relevant to the query. We show that, in this way, answers are retrieved in order of relevance and can be then returned as soon as they are built. The approach does not require the materialization of ad-hoc data structures and avoids the execution of unnecessary queries. A comprehensive evaluation demonstrates that our solution strongly reduces the complexity of the process and guarantees, at the same time, an high level of accuracy.

## 1. INTRODUCTION

Today, everyone can access an incredibly large quantity of information and this requires to rethink the traditional methods and techniques for querying and retrieving data, because the vast majority of users has little or no familiarity with computer technology. This need has originated a large set of proposals of non-conventional methods for accessing structured and semi-structured data. Among them, several studies have focused on the adoption of a keyword-based strategy for retrieving information stored in relational databases, with the goal of freeing the users from the knowledge of query languages and/or the organization of data [12, 13, 15].

EXAMPLE 1. *Let us consider the relational database in Figure 1 in which employees with different skills and responsibilities work in projects of an organization. A keyword-*

$R_1$ : Employee			$R_2$ : WorksIn	
	<u>ename</u>	<u>department</u>	<u>employee</u>	<u>project</u>
$t_1$	Zuckerberg	CS	$t_5$	Zuckerberg x123
$t_2$	Brown	CS	$t_6$	Brown cs34
$t_3$	Lee	CS	$t_7$	Lee cs34
$t_4$	Ferrucci	IE	$t_8$	Ferrucci m111

$R_3$ : Project			
	<u>id</u>	<u>pname</u>	<u>leader</u>
$t_9$	x123	Facebook	Zuckerberg
$t_{10}$	cs34	Watson	Ferrucci
$t_{11}$	ee67	LOD	Lee
$t_{12}$	m111	DeepQA	Ferrucci

$R_4$ : SkilledIn			$R_5$ : Skill	
	<u>person</u>	<u>skill</u>	<u>sname</u>	<u>type</u>
$t_{13}$	Brown	Algorithms	$t_{15}$	Algorithms theoretical
$t_{14}$	Lee	Java	$t_{16}$	Java technical

Figure 1: An example of relational database: schema and its data

based query over this database searching for experts of Java in the CS department could simply be:  $Q_1 = \{Java, CS\}$ . A possible answer to  $Q_1$  is the set of joining tuples  $\{t_3, t_{14}, t_{16}\}$ , which involve the given keywords.

Usually, keyword-based search systems over relational data involve the following key steps: (i) generation of tree-shaped answers (commonly called *joining tuple trees* or *JTT*) built by joining the tuples whose values match the input keywords, (ii) ranking of the answers according to some relevance criteria, and (iii) only the top-k answers are selected and returned to the users. The core problem of this approach is the construction of the JTT's. In this respect, the various approaches proposed in the literature can be classified in two different categories: *schema-based* [2, 16, 17, 19] and *schema-free* [13, 14, 6]. Schema-based approaches usually implement a middleware layer in which: first, the portion of the database that is relevant for the query is identified, and then, using the database schema and the constraints, a (possibly large) number of SQL statements is generated to retrieve the tuples matching the keywords of the query. Conversely, schema-free approaches first build an in-memory, graph-based, representation of the database, and then exploit graph-based algorithms and graph exploration techniques to select the subgraphs that connect nodes matching the keywords of the query.

In this paper, we present a novel technique to keyword-based search over relational databases that, taking inspira-

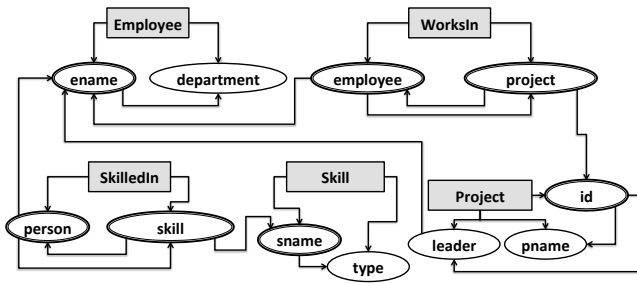


Figure 2: An example of schema graph  $SG$

tion from both the schema-based and the schema-free approaches, aims at generating progressively the most relevant answers, avoiding the selection of bunches of potential answers followed by their ranking, as it happens in other approaches. A relevant feature of our approach is that, as suggested in [18], it exploits only the capabilities of the underlying RDBMS and does not require the construction and maintenance of ad-hoc, in-memory data structures. Moreover, by avoiding redundant accesses to data, we are able to keep the computational complexity of the overall process linear in the size of the database. In a graph-oriented vision of the database, the basic idea is to search and combine incrementally the shortest paths of joining tuples that are relevant to the query. This is done by first identifying all the paths in the relational schema involving attributes linked by primary and foreign keys. Then, without building in-memory graph-shaped structures, such paths are enriched with data by traversing them backward. This step only requires simple selection and projection operations. If the backward navigation is not able to generate an answer, the paths are navigated forward using all the information retrieved in the backward phase, without further accessing the database. We show that, in this way, answers are retrieved in order of relevance. This eliminates the need to compare answers and allows us to return the results to the user as soon as they are built.

To validate our approach, we have developed a tool for keyword-based search over relational databases that implements the technique described in this paper. This tool has been used to perform several experiments over an available benchmark [4] that have shown a marked improvement over other approaches in terms of both effectiveness and efficiency.

The rest of the paper is organized as follows. Section 2 introduces a graph-based data model that we use throughout the paper. In Section 3, we describe in detail our incremental method for building top-k answers to keyword-based queries. The experimental results are reported in Section 4 and, in Section 5, we discuss related works. Finally, in Section 6, we sketch conclusions and future work.

## 2. PRELIMINARIES

### 2.1 A graph data model over relational data

In our approach, we model a relational database  $\mathbf{d}$  in terms of a pair of graphs  $\langle SG, DG \rangle$  representing the schema and the instance of  $\mathbf{d}$ , respectively. We point out however that only  $SG$  will be materialized while  $DG$  is just a conceptual notion.

DEFINITION 1 (SCHEMA GRAPH). *Given a relational*

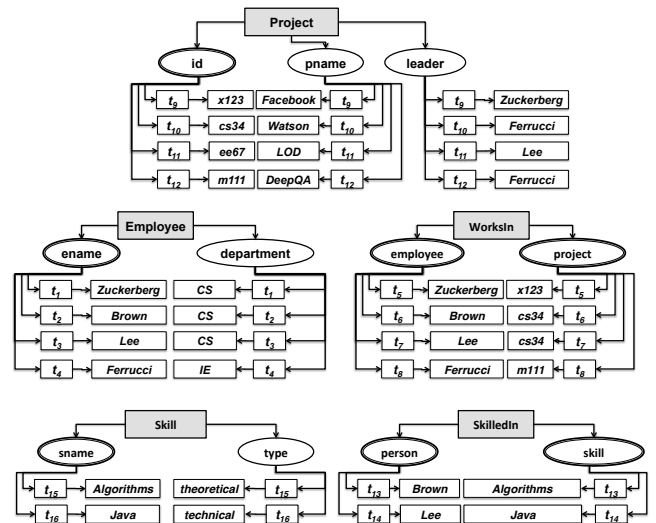


Figure 3: An example of data graph  $DG$

schema  $\mathcal{RS} = \langle \mathcal{R}, \mathcal{A} \rangle$ , where  $\mathcal{R}$  is a set of relation schemas and  $\mathcal{A}$  is the union of all attributes of  $\mathcal{R}$ , a schema graph  $SG$  for  $\mathcal{RS}$  is a directed graph  $\langle V, E \rangle$  where  $V = \mathcal{R} \cup \mathcal{A}$  and there is an edge  $(v_1, v_2) \in E$  if one of the following holds: (i)  $v_1 \in \mathcal{R}$  and  $v_2$  is an attribute of  $v_1$ , (ii)  $v_1 \in \mathcal{A}$  belongs to a key of a relation  $R \in \mathcal{R}$  and  $v_2$  is an attribute of  $R$ , (iii)  $v_1 \in \mathcal{A}$ ,  $v_2 \in \mathcal{A}$  and there is a foreign key between  $v_1$  and  $v_2$ .

For instance, the schema graph for the relational database in Figure 1 is reported in Figure 2. In a schema graph the sources represent the tables of a relational database schema (grey nodes) and the paths represent the relationships between attributes according to primary and foreign keys. The double-marked nodes denote the keys of a relation.

DEFINITION 2 (SCHEMA PATH). *A schema path in a schema graph  $SG = \{V, E\}$  is a sequence  $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_f$  where  $(v_i, v_{i+1}) \in E$  and  $v_1$  is a relation node.*

An example of schema path for the schema graph in Figure 2 is  $SkilledIn \rightarrow skill \rightarrow sname$ .

Let us now fix an injective function denoted by  $idx$  that maps each tuple to a *tuple-id* (tid for short).

DEFINITION 3 (DATA GRAPH). *Given a relational database instance  $\mathcal{I} = \langle \mathcal{R}, \mathcal{A}, I, \mathcal{D} \rangle$ , where  $I$  is the set of all tids and  $\mathcal{D}$  is the set of all data values occurring in the database, a data graph  $DG$  on  $\mathcal{I}$  is a directed graph  $\langle V, E \rangle$  where  $V = \mathcal{R} \cup \mathcal{A} \cup I \cup \mathcal{D}$  and there is an edge  $(v_1, v_2) \in E$  if one of the following holds: (i)  $v_1 \in \mathcal{R}$  and  $v_2$  is an attribute of  $v_1$ , (ii)  $v_1 \in \mathcal{A}$  belongs to a key of a relation  $R$  and  $v_2$  is the tid of a tuple for  $R$ , (iii)  $v_1$  is a tid in  $I$  and  $v_2$  is a value of a tuple  $t$  such that  $v_1 = idx(t)$ .*

Figure 3 shows the data graph on the database of Figure 1. Note that we assume, for the sake of simplicity, that each relation has an explicit attribute for its tids.

We now introduce the notion of data path. Intuitively, while a schema path represents a route to navigate relational data for query answering, a data path represents an actual navigation through data to retrieve the answer of a query.

$$\begin{aligned}
& [cl_{Java}] : \\
& \left( \begin{array}{l} dp_1 : \text{SkilledIn} \rightarrow \text{SkilledIn.skill} \rightarrow t_{14} \rightarrow \text{Java} \\ dp_2 : \text{Skill} \rightarrow \text{Skill.sname} \rightarrow t_{16} \rightarrow \text{Java} \\ dp_3 : \text{SkilledIn} \rightarrow \text{SkilledIn.person} \rightarrow x_1 \rightarrow \text{SkilledIn.skill} \rightarrow t_{14} \rightarrow \text{Java} \\ dp_4 : \text{SkilledIn} \rightarrow \text{SkilledIn.skill} \rightarrow x_2 \rightarrow \text{Skill.sname} \rightarrow t_{16} \rightarrow \text{Java} \\ dp_5 : \text{SkilledIn} \rightarrow \text{SkilledIn.person} \rightarrow x_3 \rightarrow \text{SkilledIn.skill} \rightarrow x_4 \rightarrow \text{Skill.sname} \rightarrow t_{16} \rightarrow \text{Java} \end{array} \right) \\
& [cl_{CS}] : \\
& \left( \begin{array}{l} dp_6 : \text{Employee} \rightarrow \text{Employee.department} \rightarrow t_1 \rightarrow CS \\ dp_7 : \text{Employee} \rightarrow \text{Employee.department} \rightarrow t_2 \rightarrow CS \\ dp_8 : \text{Employee} \rightarrow \text{Employee.department} \rightarrow t_3 \rightarrow CS \\ \dots \\ dp_9 : \text{SkilledIn} \rightarrow \text{SkilledIn.person} \rightarrow x_5 \rightarrow \text{Employee.ename} \rightarrow x_6 \rightarrow \text{Employee.department} \rightarrow t_3 \rightarrow CS \\ \dots \end{array} \right)
\end{aligned}$$

Figure 4: Clusters of data paths for  $Q_1 = \{\text{Java}, \text{CS}\}$

DEFINITION 4 (DATA PATH). Given a schema path  $sp = R \rightarrow A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_k$  the data path  $dp$  following  $sp$  is the path  $R \rightarrow A_1 \rightarrow \tau_1 \rightarrow \dots \rightarrow A_k \rightarrow \tau_k \rightarrow v$ , where: (i) each  $\tau_i$  denotes either a variable denoting a tid or the tid of a tuple belonging to the relation involving  $A_i$  and (ii)  $v$  is a value belonging to the tuple with tid  $\tau_k$ .

Let us consider again the example in Figure 3. The data path that follows the schema path  $sp = \text{SkilledIn} \rightarrow \text{skill} \rightarrow \text{sname}$  is the following:

$$dp_1 : \text{SkilledIn} \rightarrow \text{skill} \rightarrow x_1 \rightarrow \text{sname} \rightarrow t_{15} \rightarrow \text{Algorithms}$$

Basically, this path describes the fact that the *sname* of the tuple with tid  $t_{15}$  is related to the *skill* of a tuple  $x_1$  in relation *SkilledIn*.

An instance of a data path  $dp$  is a function  $\phi$  that associates a tid with each variable occurring in  $dp$ . As an example, an instance of the data path  $dp_1$  above associates  $t_{13}$  with  $x_1$ .

## 2.2 Answers to a keyword-based query

We consider the traditional Information Retrieval approach to value matching adopted in full text search and we denote the matching relationship between values with  $\approx$ . We have used standard libraries for its implementation and since this aspect is not central in our approach, it will not be discussed further. Given a tuple  $t$  and a value  $v$ , we then say that  $t$  matches  $v$ , also denoted for simplicity by  $t \approx v$ , if there is a value  $v'$  in  $t$  such that  $v \approx v'$ .

DEFINITION 5 (ANSWER). An answer to a keyword-based query  $Q$  is a set of tuples  $S$  such that: (i) for each keyword  $q$  of  $Q$  there exists a tuple  $t$  in  $S$  that matches  $q$  and (ii) the tids of the tuples in  $S$  occur in a set of data path instances having at least one tid in common.

An example of answer, with reference to the query  $Q_1 = \{\text{Java}, \text{CS}\}$ , is the set of tids  $\{t_3, t_{14}, t_{16}\}$  that are contained in the instances of the set  $\{dp_5, dp_9\}$  of data path in Figure 4.

Note that we assume the AND semantics for the keywords in  $Q$ . Note also that our notion of answer basically corresponds to the notion of joining tuple tree (JTT) [11].

As usual, an answer  $S_1$  is considered more relevant than another answer  $S_2$  if  $S_1$  is “more compact” than  $S_2$  since, in this case, the keywords of the query are closer between each other [5]. This is captured by a scoring function that simply returns cardinality of  $S$ .

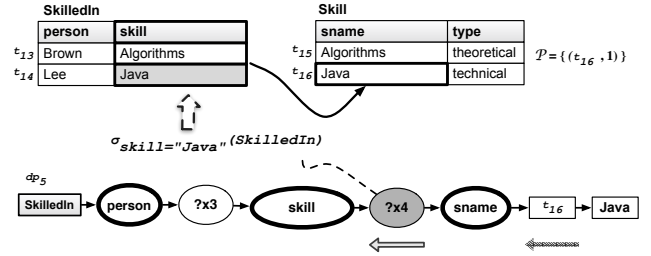


Figure 5: Backward exploration at work for  $dp_5$  (selection)

**Problem Statement.** Given a relational database  $\mathbf{d}$  and a keyword search query  $Q = \{q_1, q_2, \dots, q_{|Q|}\}$ , where each  $q_i$  is a keyword, we aim at finding the top- $k$  ranked answers  $S_1, S_2, \dots, S_k$ .

## 3. PATH-ORIENTED SEARCH

Given a keyword-based query  $Q$ , our technique consists of two main phases, *clustering* and *building*. They guarantee a monotonic construction of the answers (i.e. the answer generated in the  $i$ -th step is always more relevant than that of the  $i + 1$ -th step) and a linear time complexity with respect to the size of the input. This makes possible to return answers as soon as they are computed.

### 3.1 Clustering

In the first phase all the data paths having an ending node that matches one of the keywords in  $Q$  are generated and grouped in clusters. There is one cluster for each keyword  $q_i \in Q$ . In particular, we start from each data path  $R \rightarrow A \rightarrow tid \rightarrow v$  such that  $q_i \approx v$ . Then we generate the data paths following the route of each schema path  $sp$  ending into the attribute  $A$ . The clusters are kept ordered according to the length of the data paths, with the shortest paths coming first. As an example, given the query  $Q_1 = \{\text{Java}, \text{CS}\}$  and the relational database in Figure 1, we obtain the clusters shown in Figure 4.

### 3.2 Building

The second phase aims at generating the most relevant answers by combining the data paths generated in the first step. This is done iteratively by picking, in each step, the shortest data paths from each cluster: if there is an instance of these data paths having a tuple in common, we have found



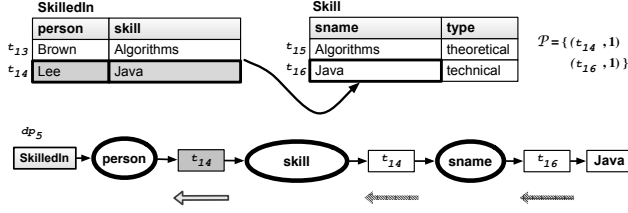


Figure 6: Backward exploration at work for  $dp_5$  (projection)

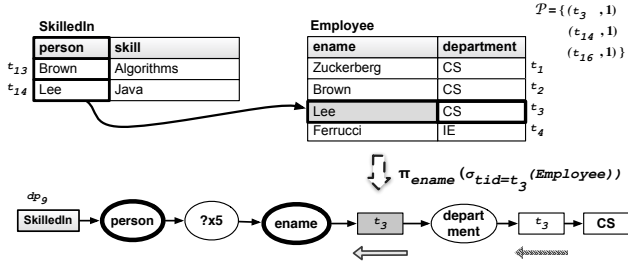


Figure 7: Backward exploration at work for  $dp_9$  (projection)

an answer. The search proceeds in this way with longer data paths that follow in the clusters. In detail, following the Algorithm 1, we extract all top data paths (i.e. the shortest ones) from each cluster into a set  $DP$  (lines 5-6). This task is supported by the procedure `dequeueTop`. Then we generate all possible combinations  $C$  of paths within  $DP$  (line 12) in order to find the best candidates to be answers (i.e. the task is performed by the procedure `validCombinations`). Each combination  $c$  is a connected directed graph that has to contain exactly one data path from each cluster: two paths from the same cluster cannot belong to the same combination and all clusters have to participate in each combination, i.e. AND-semantics of answers. We try to combine paths with the same length. However two clusters could provide their longest paths with different length. In this case, to satisfy the AND-semantics, if a cluster  $cl_i$  becomes empty then we re-enqueue those data paths  $dp \in DP$  such that  $dp \triangleright cl_i$  (lines 9-11). Note that, given a cluster  $cl_i$  corresponding to a keyword  $q_i$ , if  $q_i \approx \text{last}(dp)$  then we denote  $dp \triangleright cl_i$ . In this case we combine also data paths with different length.

For instance referring to our example with the clusters in Figure 2, at the first running of the algorithm we have to combine  $dp_1, dp_2$  from  $cl_{Java}$  with  $dp_6, dp_7, dp_8$  from  $cl_{CS}$ . To avoid a possible exponential number of combinations and useless path processing, we check, through the procedure `validCombinations`, before combining paths if all those paths cross a common tid table. This is a necessary condition for finding a common tid node. Intuitively the best answer contains tuples strictly correlated, e.g., a tuple containing all the keywords or tuples directly correlated by foreign key constraints.

Referring to our example there is no valid combination in the first two runs of the algorithm. Therefore we have to extract longer data paths from  $\mathcal{CL}$  and we find the first valid combination that is  $c = \{dp_5, dp_9\}$ . Now we have to verify if  $c$  brings an answer: if the test is positive, we extract all tids of  $c$ , i.e. the answer  $S_i$ , to include in the

### Algorithm 1: Building

---

**Input** : The clusters  $\mathcal{CL}$ , a query  $Q$ , the number  $k$ .  
**Output**: The set of answers  $S$ .

---

```

1 finished ← false;
2 S ← ∅;
3 while ¬finished do
4   DP ← ∅;
5   foreach  $cl_i \in \mathcal{CL}$  do
6     DP ← DP ∪ dequeueTop( $cl_i$ );
7   if  $\mathcal{CL} = \emptyset$  then finished ← true;
8   else
9     foreach  $cl_i \in \mathcal{CL}: cl_i = \emptyset$  do
10      foreach  $dp \in DP: dp \triangleright cl_i$  do
11         $cl_i.enqueue(dp)$ ;
12   C ← validCombinations(DP);
13   foreach  $c \in C$  do
14     P ← ∅;  $C_d \leftarrow \emptyset$ ;
15     foreach  $dp \in c$  do
16        $is\_sol \leftarrow$ 
17       backward_exploration( $dp, Q, P, C_d$ );
18       if  $is\_sol$  then
19         S.enqueue(P.keys);
20       else if forward_exploration(P,  $C_d$ ) then
21         S.enqueue(P.keys);
22       if |S| = k then
23         return S;

```

---

set  $S$ . This means to instantiate a set set of data paths  $DP = \{dp_1, \dots, dp_n\}$  and verifying if the results have a tuple in common. This evaluation is performed by the procedures `backward_exploration` and `forward_exploration`, as follows. Such procedures keep a map  $\mathcal{P}$  where the key is a tid and the value is the number of occurrences of the tid in the combination  $c$ . If  $c$  brings an answer, then  $S_i$  is the set of keys extracted from  $\mathcal{P}$  (line 18 and line 20). The building ends when we computed  $k$  answers (line 22) or the set  $\mathcal{CL}$  is empty (line 7).

**Backward Exploration.** Each data path  $dp$  of a combination is analysed independently from the others, i.e. in our example  $dp_5$  and  $dp_9$ . They are navigated backward starting from the last node. In other terms we follow the

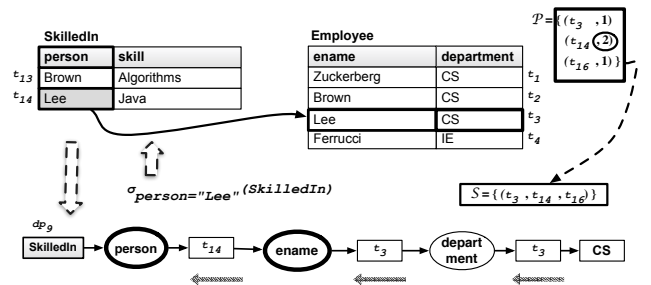


Figure 8: Backward exploration at work for  $dp_9$  (selection)

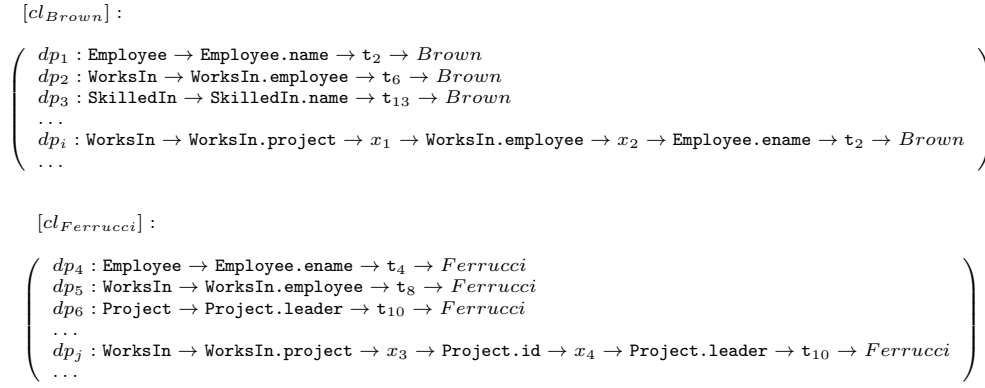


Figure 9: Clusters of data paths for  $Q_2 = \{Brown, Ferrucci\}$

foreign and primary key constraints contrariwise. The `backward_exploration` procedure takes as input a data path  $dp$  to analyse, the query  $Q$ , a map  $\mathcal{P}$  and a set  $\mathcal{C}_d$  of conditions, whose functionality will be described in the forward exploration. Given  $dp_5$ , we start from the node *Java*, we meet the tid  $t_{16}$  and the algorithm updates  $\mathcal{P}$  inserting the pair  $\{t_{16}, 1\}$ . Then, we proceed until the variable  $x_4$  is encountered (Figure 5).

According to the information carried by this data path, the only possible substitution for  $x_4$  is  $t_{14}$ , that is the tid of the tuple that has as `SkilledIn.skill` the same value occurring in `Skill.sname` of the tuple with tid  $t_{16}$ , i.e. *Java*. In this case we are following a foreign key constraint and we extract the new tid by a simple selection. As shown in Figure 6, it turns out that  $x_3 = t_{14}$  as well, since  $x_4$  and  $x_3$  refer to the same tuple in the `SkilledIn` relation. In this case we are following a primary key constraint and the procedure extracts the data value associated to the attribute  $A$  of the same tuple with a simple projection. The exploration of  $dp_5$  terminates. Similarly we explore  $dp_9$ . In Figure 7, we start from the data value *CS*, we insert  $t_3$  in  $\mathcal{P}$  and then we meet the variable  $x_6$ . It belongs to the same relation `Employee` of  $t_3$ . Therefore  $x_6$  corresponds to  $t_3$  and it is extracted by a projection.

Finally, we meet the variable  $x_5$  as depicted in Figure 8. Since we are following a foreign key constraint, we execute the selection  $\sigma_{person="Lee"}(\text{SkilledIn})$  and we retrieve the tid  $t_{14}$ . In this case  $t_{14}$  exists in  $\mathcal{P}$ : we have to increment the value associated to  $t_{14}$  in  $\mathcal{P}$ . If  $\mathcal{P}$  contains a pair  $\{t, n\}$ , where  $n = |Q|$ , then  $t_y$  represents the tuple able to reach all tuples matching the keywords of  $Q$ : in this case the tids in  $\mathcal{P}$  represent an answer to insert in  $\mathcal{S}$ ; in our example we have the answer  $\{t_3, t_{14}, t_{16}\}$ .

**Forward Exploration.** If in the backward exploration we find a multiple substitutions for some variable the analysis of the current data path stops. In this case we would need to fork the exploration for each retrieved result: we could trigger a large number of branches and consequently explore all the database  $\mathbf{d}$  more times, similarly to schema free approaches.

Therefore, the backward exploration determines a condition  $\gamma$  in terms of a triple  $\langle R, A, v \rangle$ . The condition says that a tuple in the relation  $R$  having the data value  $v$  associated to the attribute  $A$  is desired. All the conditions are kept in a set  $\mathcal{C}_d$ . Starting from the information captured by the conditions we use a *forward* strategy, where data paths are

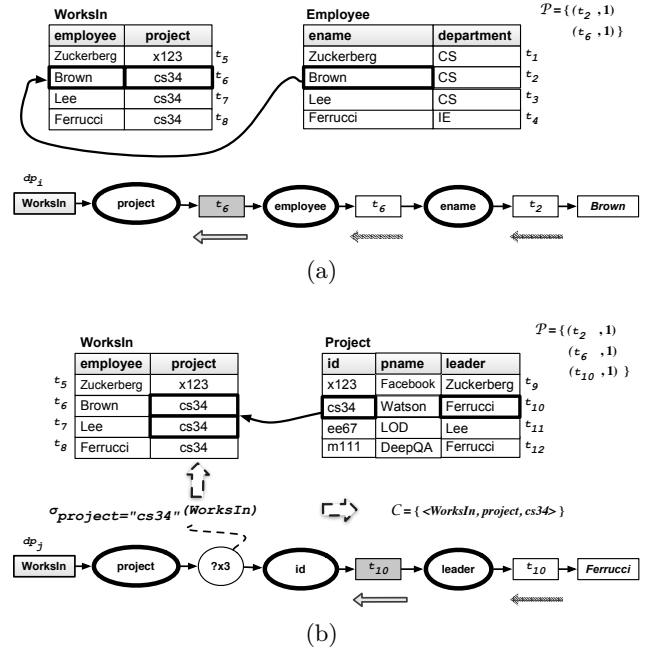


Figure 10: Backward exploration at work for  $Q_2$

navigated forward using all the tids retrieved in the first step as substitutions for the remaining variables.

For instance, let us consider a second query  $Q_2 = \{Brown, Ferrucci\}$ . In this case we would retrieve information about *Brown* and how he is related to *Ferrucci*. We obtain the two clusters depicted in Figure 9. In this case the first desired answer should be  $S_1 = \{t_2, t_6, t_{10}\}$ , i.e. *Brown* works in the *CS* department and he works in the *Watson* project with id *cs34* whose director is *Ferrucci*. In Figure 10 we depict the backward exploration at work to process the query  $Q_2$ .

The first combination  $c$  useful to generate  $S_1$  is  $(dp_i, dp_j)$ . However, in this case the backward exploration is not able to provide an answer. Through the selection

$$\sigma_{\text{employee}="Brown"}(\text{WorksIn})$$

it is possible to instantiate the variables  $x_1$  and  $x_2$  with  $t_6$  in  $dp_i$ , as shown in Figure 10.(a). In  $dp_j$  the variable  $x_4$  is trivially instantiated with  $t_{10}$ , but the procedure stops when it tries to resolve the variable  $x_3$ . This is due to perform the



selection  $\sigma_{project="cs34"}(WorksIn)$ , resulting more than one tid, i.e.  $t_6$  and  $t_7$ . At the end of the backward exploration we have  $\mathcal{P} = \{(t_2, 1), (t_6, 1), (t_{10}, 1)\}$  and the condition  $\gamma_1 = \langle WorksIn, project, "cs34" \rangle$  in the set  $\mathcal{C}_d$ . To retrieve  $S_1$  the forward exploration has to disambiguate between  $t_6$  and  $t_7$ . Since  $t_7$  is not in  $\mathcal{P}$ , we do not consider it. Incrementing the value associated to  $t_6$  in  $\mathcal{P}$  we obtain the pair  $(t_6, 2)$ , i.e. we find the first answer  $S_1 = \{t_2, t_6, t_{10}\}$ .

In general, the *projection* step could fail: a single condition  $\gamma$  is not able to disambiguate tuples, i.e.  $\nexists t_y \in \mathcal{P} : t_y \models \gamma$ . In this case we have to retrieve new tids from  $\mathbf{d}$ . Therefore the forward exploration provides the *selection* step. In  $\mathcal{C}_d$ , we search multiple conditions involving the same relation  $R$ , i.e.  $\gamma_1 = \langle R, A_1, v_1 \rangle, \gamma_2 = \langle R, A_2, v_2 \rangle, \dots, \gamma_n = \langle R, A_n, v_n \rangle$ , and then we check if these multiple conditions can retrieve a new tid  $t_y$  in  $R$ .

Note that the forward navigation has been already exploited in data graph algorithms [9, 12] to improve backward explorations individuating connections from potential root nodes to keyword nodes. Similarly, our forward exploration supports the backward navigation, still preserving our competitive advantages: it does not require to keep extra information of the exploration and it only exploits *selection* ( $\sigma$ ) and *projection* ( $\pi$ ) operations.

## 4. EXPERIMENTAL RESULTS

We developed our approach in YAANIIR, a system for keyword search over relational databases. YAANIIR is implemented entirely with a procedural language for SQL. In particular PL/pgSQL since we used PostgreSQL 9.1 as RDBMS. In our experiments we used the only available benchmark, which is provided by Coffman et al. [4]. It satisfies criteria and issues [3, 20] from the research community to standardize the evaluation of keyword search techniques. In [4], by comparing the state-of-the-art keyword search systems, the authors provide a standardized evaluation on three datasets of different size and complexity: IMDB (1,67 million tuples and 6 relations), WIKIPEDIA (206.318 tuples and 6 relations), and a third ideal counterpoint (due to its smaller size), MONDIAL (17.115 tuples and 28 relations). For each dataset, we run the set of 50 queries (see [4] for details and statistics). Experiments were conducted on a dual core 2.66GHz Intel Xeon, running Linux RedHat, with 4 GB of memory, 6 MB cache, and a 2-disk 1TB striped RAID array, and we used PostgreSQL 9.1 as RDBMS. We remark that we keep schema and instance of all datasets.

**Implementation.** The implementation plays an essential role in our framework. Here we provide some technical details in order to show the feasibility to implement keyword-based search functionality in a RDBMS and consequently to introduce an SQL keyword search operator. We implemented the algorithms of the paper by using only a procedural language for SQL and the RDBMS data structures. Similarly to all the approaches we employ inverted indices and full-text queries to have direct access to the tuples of interest. Modern RDBMSs already integrate general purpose full-text indices and related query operators. In some case they can be customized by the DB administrator and applied on a limited number of attributes, i.e. usually the attributes relevant to the user or containing text data. We implement schema and data paths as integer arrays, i.e. text values are encoded by hash functions provided by the RDBMS. Each element of the array corresponds to a node

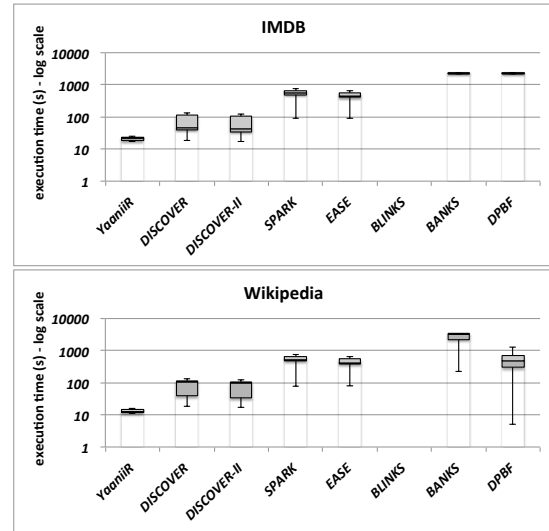
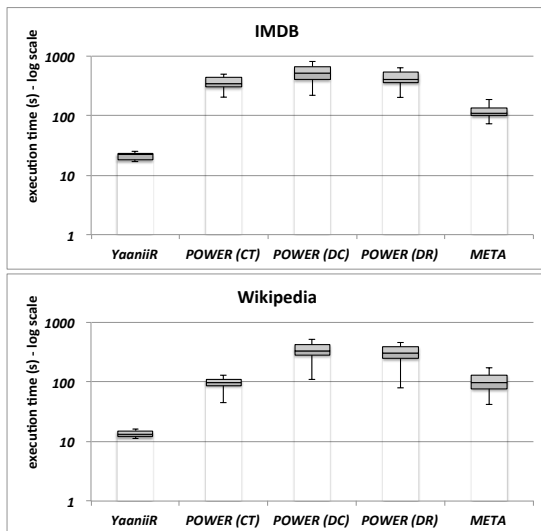


Figure 11: Performance comparison with schema-free approaches

in the path. Schema paths are retrieved by the computation of the metadata (schema) of  $\mathbf{d}$ . The management of tuple-ids is already implemented in many RDBMSs. In our case, we use the PostgreSQL clause `WITH OIDS` updating the definition of a table, in case. It creates a column named `OID` containing the identifiers of the tuples. Each cluster is in practice a priority queue where the priority decreases with the increasing length of a path. A cluster is implemented with a table, having the length of the paths as indexed attribute. All the loops of the algorithms are supported by the definition and usage of cursors. In our implementation we apply a straightforward cache mechanism for the tuples. In the cache we trace the already accessed tuples. So before executing an access to the disk we search within the cache. In this way a tuple is accessed only once. Such simple mechanism speeds-up significantly the execution time.

Our algorithms have been implemented in terms of PL/pgSQL procedures to add in  $\mathbf{d}$ . Such procedures exploit a simple index based on the permanent table  $SG(attribute, path)$  and the procedure `DG`. The former stores all schema paths while the latter retrieve all data paths at runtime. In `SG`, `path` implements a schema path in terms of an array of hash numbers (i.e. hashing of table and attributes names in the schema) while `attribute` is the value of the ending node of the path implemented as a hash value (i.e. on `attribute` we define a B-tree index). An efficient implementation of a BFS traversal supports the computation of all schema paths (i.e. we compute all paths between tables and attributes, not only the shortest ones). The procedure `DG`, similarly, implements a data path in terms of an array of hash numbers and defines a `tsvector` value on all text attributes of  $\mathbf{d}$  on which imposes a GIN index for full-text search. Such pre-configuration (e.g., the building of the `SG` table) is built efficiently: from few milliseconds on `MONDIAL` to a couple of minutes on `IMDB` and `WIKIPEDIA`. The last datasets, i.e. `IMDB` and `WIKIPEDIA`, present 516MB and 550 MB of size, respectively. The resulting index increases the starting data size of few MBs.

**Performance.** For query execution evaluation, we compared our system (YAANIIR), with the most related



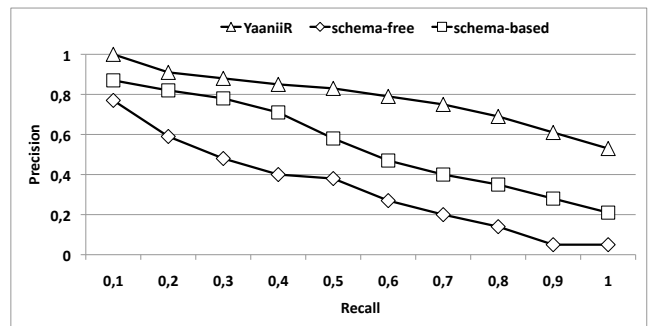
**Figure 12: Performance comparison with schema-based approaches**

schema-free approaches: SPARK [16], EASE [14], and BLINKS [9], DPBF [7], DISCOVER [11] and the refined version DISCOVER-II [10]. Moreover we made a comparison with schema-based approaches: POWER [18], using all the algorithms under the three semantics – connected tree (CT), distinct core semantics (DC), distinct root semantics (DR)<sup>1</sup> – and META [2].

We evaluated the execution time that is the time elapsed from issuing a query until an algorithm terminates. Such execution computes the top-100 answers. We performed *cold-cache* experiments (by dropping all file-system caches before restarting the systems and running the queries) and *warm-cache* experiments (without dropping the caches). We repeated all the tests three times and measured the mean execution times. For space constraints, we report only cold-cache experiments, but warm-cache experiments follow a similar trend. As in [4], we imposed a maximum execution time of 1 hour for each technique (stopping the execution and denoting a timeout exception). Moreover we allowed  $\approx 5$  GB of virtual memory and limit the size of answers to 5 tuples.

Figure 11 and Figure 12 show box plots of the execution times for all queries on each dataset w.r.t schema-free approaches and schema-based approaches, respectively. In general our system outperforms consistently all approaches. In particular the range in execution times for schema-free approaches is often several orders of magnitude: the performance of these heuristics varies considerably (i.e. the evaluation of the mean execution time cannot report such behavior). In the figures, we do not report box plots for BLINKS since it always required more than one hour or encountered an `OutOfMemoryError`. Similarly, DISCOVER, BANKS, DPBF failed many queries due to time out exception. SPARK and EASE perform worse but they completed most of the queries. Our system completed all 50 queries in each dataset without computing useless answers or set of tuples to combine. This is due to our incremental strategy reducing the space overhead and consequently the time complexity of the overall process w.r.t. the competitors that

<sup>1</sup>We refer to the most efficient version of both DC and DR



**Figure 13: Precision-Recall curves**

spend much time traversing a large number of tuples (nodes) and computing and ranking the candidates to be (in case) answers.

With respect to schema-based approaches, we implemented the three algorithms of POWER in Java 1.6 and JDBC to connect to PostgreSQL. In particular we used the same parameters for IMDB testing as described in [18] for all datasets. On the other hand, we used the implementation of META offered by the same authors. Also in this case, the results confirm the significant speed-up of our approach with respect to the others. In this case the number of tuples generated by the join operations is effective to generate the answers of interest, i.e. the cost to evaluate each candidate network is limited. The DC and DR algorithms perform worse due to the more complex technique to evaluate the candidate networks. In some queries, a larger number of keywords in  $Q$  increases the complexity to evaluate a candidate network and consequently the number of tuples to evaluate. In this context the CT algorithm and META are comparable while our system performs significantly better due to the lowest (or missing) overhead introduced in our incremental strategy. However schema-based approaches completed all 50 queries in each dataset and provide a more regular behavior in the execution time.

**Effectiveness.** We have also evaluated the effectiveness of results. We measured the interpolation between precision and recall to find the top-10 answers, on the queries on all datasets. We compare our curve with the interpolated precision curves averaged over both schema-free and schema-based approaches. Figure 13 shows the results. As to be expected, the precision of the other systems dramatically decreases for large values of recall. The overhead introduced by all competitors damages the quality of the results. On the contrary our strategies keeps values on the range [0.6,0.9]. Such result confirms the discussion of Section 3, that is the feasibility of our system that produces the top-k answers in linear time.

## 5. RELATED WORK

The common assumption made by the various proposals to keyword search over relational databases is that an answer is a *joining tuple tree* (JTT) in which the nodes represent tuples and the edges represent references between them, according to the foreign keys defined on the database schema. The various approaches to keyword-based query answering are commonly classified into two categories, *schema-based* and *schema-free*, even if some recent works have questioned the state of the art and suggested alternative techniques to solve

the problem. We discuss all of them in order.

**Schema-based approaches.** Schema-based approaches [11, 16, 17] make use, in a preliminary phase, of the database schema to build trees called *candidate networks* (CNs) whose nodes represent subsets of the tuples in a relation. CNs must be *complete* (i.e., involving all the keywords in the query) and *duplicate-free*. Duplicate elimination relies on graph isomorphisms, which requires a high computational cost. For this reason, in [17] the authors have proposed an approach to CN duplicate elimination that does not rely on graph isomorphism. CNs are then evaluated by means of a (possible large) number of SQL queries that, once submitted to the RDBMS, return the final JTTs. Unfortunately, it has been shown that finding the best execution plan from a set of CNs is an NP-Complete problem [11]. Moreover, empty results can occur and this can make the process inefficient and introduce noise in the final result. Our approach fits in this category in that we take advantage from database schema and constraints to build the data paths (see Definition 4) without accessing the database.

**Schema-free approaches.** Schema-free approaches [6, 7, 13, 14] first build a graph-based representation  $\mathcal{G}$  of the database in which the nodes of  $\mathcal{G}$  represent the tuples of the database and its edges represent primary or foreign key constraints. Then, they make use of graph algorithms and graph exploration techniques to select the subgraphs of  $\mathcal{G}$  that connect nodes matching the keywords of the query. Usually, apart from [6], all of them materialize  $\mathcal{G}$  in main memory, which is clearly hard to scale. Query evaluation usually consists in finding a set of (minimal) *Steiner trees* [8] of  $\mathcal{G}$ . This problem is known to be NP-Complete [8]. Therefore, the various proposals rely on complex heuristics aimed at generating approximations of Steiner trees. We actually took inspiration from these approaches by modeling the problem in terms of graph search. However, we do not build in-memory graph-based structures and resort on a simple technique for building the answers that is linear in the size of the database and does not require complex graph algorithms of high computational cost.

**New approaches.** As observed by several authors (e.g., [1, 4]), the solutions proposed so far are not efficient and reliable enough for a spread usage. Indeed, it should be mentioned that none of them has been implemented in a commercial system. The authors in [18] argue that the main drawback of existing approaches is the limited use of the functionality of the RDBMS in which data is stored. The work in [1] proposes to compute the answers within a time limit and to show to the user the unexplored part of the database, so that she can refine the results. We have indeed followed this clue in that our approach only relies on the capabilities of the underlying RDBMS.

## 6. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel approach to keyword search query over relational databases, by providing a linear strategy for top-k query answering. Such strategy enables the search to scale seamlessly with the size of the input. Experimental results confirmed our algorithms and the advantage over other approaches. This work now opens several directions of further research. From a theoretical point of view, we are investigating algorithms to keyword search over

distributed environments, retaining the results achieved in this paper. From a practical point of view, we are widening optimization techniques to speed-up the query evaluation and to improve the effectiveness of the result, implementing an SQL operator.

## 7. REFERENCES

- [1] A. Baid, I. Rae, J. Li, A. Doan, and J. F. Naughton. Toward scalable keyword search over relational data. *PVLDB*, 3(1):140–149, 2010.
- [2] S. Bergamaschi, E. Domnori, F. Guerra, R. T. Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, pages 565–576, 2011.
- [3] Y. Chen, W. W. 0011, Z. Liu, and X. Lin. Keyword search on structured and semi-structured data. In *SIGMOD*, pages 1005–1010, 2009.
- [4] J. Coffman and A. Weaver. An empirical performance evaluation of relational keyword search techniques. *TKDE*, 99(PrePrints):1, 2012.
- [5] J. Coffman and A. C. Weaver. Learning to rank results in relational keyword search. In *CIKM*, pages 1689–1698, 2011.
- [6] B. B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *VLDB*, 1(1):1189–1204, 2008.
- [7] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.
- [8] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977.
- [9] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.
- [10] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.
- [11] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.
- [12] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.
- [13] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pages 173–182, 2006.
- [14] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.
- [15] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.
- [16] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, pages 115–126, 2007.
- [17] A. Markowetz, Y. Yang, and D. Papadias. Keyword search on relational data streams. In *SIGMOD*, pages 605–616, 2007.
- [18] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of RDBMS. In *SIGMOD*, pages 681–694, 2009.
- [19] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.
- [20] W. Webber. Evaluating the effectiveness of keyword search. *IEEE Data Eng. Bull.*, 33(1):54–59, 2010.

# Implementing Iterative Algorithms with SPARQL

Robert W. Techentin,  
Barry K. Gilbert  
Mayo Clinic  
Rochester, MN  
{techentin,robert,  
gilbert.barry}@mayo.edu

Adam Lugowski, Kevin  
Deweese, John Gilbert  
UC Santa Barbara  
Santa Barbara, CA  
{alugowski,kdeweese,  
gilbert}@cs.ucsb.edu

Eric Dull, Mike Hinchey,  
Steven P. Reinhardt  
YarcData LLC  
Pleasanton, CA  
{edull,mhinchey,spr}  
@yarcdata.com

## ABSTRACT

The SPARQL declarative query language includes innovative capabilities to match subgraph patterns within a semantic graph database, providing a powerful base upon which to implement complex graph algorithms for very large data. Iterative algorithms are useful in a wide variety of domains, in particular in the data-mining and machine-learning domains relevant to graph analytics. In this paper we describe a general mechanism for implementing iterative algorithms via SPARQL queries, illustrate that mechanism with implementation of three algorithms (peer-pressure clustering, graph diffusion, and label propagation) that are valuable for graph analytics, and observe the strengths and weaknesses of this approach. We find that writing iterative algorithms in this style is straightforward to implement, with scalability to very large data and good performance.

## Keywords

graph analysis, SPARQL, data mining, iterative algorithms, clustering, query languages, performance

## 1. OVERVIEW

The SPARQL declarative query language [7] implements innovative capabilities to match subgraph patterns within a semantic graph database, providing a powerful base upon which to implement complex graph algorithms for very large semantic (or heterogeneous) data. SPARQL has major advantages for practical problem-solving, including its built-in support for semantic graph querying, its status as an emerging standard from the W3C along with its companion Resource Description Framework (RDF) [12] data format, and its implementation by numerous providers of both databases and tools, including Jena [1], Sesame [10], AllegroGraph [3], TopBraid Composer [18], and Urika [19]. The use of SPARQL is growing, so understanding its current capabilities and limitations is valuable, so it can be used to address the widest practical range of graph-analytic problems.

Iterative algorithms are useful in a wide variety of domains related to graph analytics, esp. data mining and machine learning, so having such algorithms readily implementable in SPARQL extends the range of practical algorithms considerably. We present one approach for implementing iterative algorithms in SPARQL, consisting of a) a set of initial queries that establishes a baseline state, b) a set of iterative queries that updates the state (typically via SPARQL Update constructs) and calculates the current value of convergence criteria, and c) a set of final queries that creates final results and cleans up intermediate state.

We illustrate this method via the implementation of three algorithms that calculate per-vertex metrics that depend on the structure of the graph. *Peer-pressure clustering* [15] groups vertices into clusters based on the cluster to which most of a vertex's neighbors belong. *Graph diffusion* [5] calculates the diffusion of an effect from seeded nodes throughout the graph, identifying both vertices that are likely to be related as well as pathways that contribute to the relationship. *Label propagation* [11] propagates known outcomes from a set of labeled data through a set of unlabeled data, tagging vertices with their likely outcomes based on the information latent in the graph.

We find that writing iterative algorithms in this style is straightforward to implement, with scalability to very large data and good performance. Though there are algorithms for which the iterative queries are so simple that the overhead of executing any query may be a performance issue, initially implementing such algorithms in this style delivers correct answers quickly, with an optimized implementation possible via other means if needed.

While there are other approaches to this problem, notably the work on recursive database queries with Datalog [16], our focus is on SPARQL because of its intended audience of subject-matter experts, not professional programmers.

## 2. THE SPARQL 1.1 LANGUAGE

SPARQL is a query language for semantic-graph databases containing data represented in the Resource Description Framework (RDF) [12], with its name being a recursive acronym for SPARQL Protocol and RDF Query Language. It comes from the semantic web community and is a recommendation of the World Wide Web Consortium [4]. The primary goal for RDF was to make web pages machine-readable, and the goal for SPARQL was to enable higher-level querying of the semantic web. The resulting capabilities proved to be valuable for graphs that did not necessarily originate as web pages; *i.e.*, queries on highly heterogeneous

and richly interconnected data, data that reflected the Open World Assumption [5] that one's set of data is never complete and so tools must be built expecting to easily incorporate new data and new types of data. Readers who use SQL [6] will find many SPARQL constructs familiar.

RDF defines data in terms of *triples* consisting of a subject, a predicate or relationship, and an object. For example, the triple "Ruth works-for Mayo-Clinic" has "Ruth" as the subject, "works-for" as the predicate, and "Mayo-Clinic" as the object. Well-defined RDF data will use Universal Resource Identifiers (URIs, [9]) for subjects, predicates, and most objects. An RDF graph is a collection of these triples.

An example of SPARQL graph matching comes from the Lehigh University Benchmark (LUBM) query #2 [6]: (SPARQL keywords are shown in upper case for clarity)

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-
  rdf-syntax-ns#>
2 PREFIX ub: <http://www.lehigh.edu/~zhp2
  /2004/0401/univ-bench.owl#>
3 SELECT ?student ?faculty ?course
4 WHERE {
5   ?student rdf:type ub:Student .
6   ?faculty rdf:type ub:Faculty .
7   ?course rdf:type ub:Course .
8   ?student ub:advisor ?faculty .
9   ?faculty ub:teacherOf ?course .
10  ?student ub:takesCourse ?course
11 }

```

SPARQL variables are denoted by an initial ? or \$ character, e.g., ?student in the example above. The statements within the WHERE clause, known as a *basic graph pattern*, can be interpreted as "find all triples (?student, ?faculty, ?course) where (lines 5-7) ?student, ?faculty, and ?course are of the corresponding types, and there exists (8) an edge of type ub:advisor from the ?student vertex to the ?faculty vertex, (9) an edge of type ub:teacherOf from the ?faculty vertex to the ?course vertex, and (6) an edge of type ub:takesCourse from the ?student vertex to the ?course vertex." In natural language, the query can be stated "find triples of Student, Faculty, and Course where the student takes a course taught by her advisor."

Once a graph pattern has been matched, the intermediate solution can be further processed or combined with other intermediate solutions. E.g., LUBM query #2 could be modified as follows. The inner SELECT query (lines 4-14 below) matches the basic graph pattern in the WHERE, groups those results first by ?faculty and then within a single ?faculty value by ?student; then those groups are aggregated by selecting the (unique, per group) distinct value of ?faculty and ?student and by COUNTing the instances of ?course per group, and then keeping only the results that have a ?courseCount greater than 2. The outer query takes the results of the inner query, groups them by ?faculty and COUNTs the number of students (per faculty) who have taken more than one course from their advisor. The full result is faculty members who have advisees who have taken more than one course from them, sorted in descending order of the count of such advisees per faculty member.

```

1 SELECT ?faculty
2   (COUNT(?student) AS ?studentCount)
3 WHERE {
4   SELECT ?faculty ?student
5     (COUNT(?course) AS ?courseCount)
6   WHERE {
7     ?student rdf:type ub:Student .
8     ?faculty rdf:type ub:Faculty .
9     ?course rdf:type ub:Course .
10    ?student ub:advisor ?faculty .
11    ?faculty ub:teacherOf ?course .
12    ?student ub:takesCourse ?course
13  } GROUP BY ?faculty ?student
14    HAVING (?courseCount > 1)
15 } GROUP BY ?faculty
16   ORDER BY DESC(?studentCount)

```

We note other SPARQL constructs, including *named graphs*, each of which segregates a set of triples from the main body (*default graph*) of the graph database, enabling its simple identification for a specific use. SPARQL also supports aggregate functions and mathematical operators sufficient for basic computations on query results. FILTER limits results by various function comparisons rather than graph pattern. MINUS, EXISTS, and NOT EXISTS offer different ways of reducing results by graph pattern matching. OPTIONAL is like SQL's LEFT JOIN, allowing bindings that may not be present for some results.

SPARQL 1.1 also includes a set of Update capabilities [4], including INSERT, which adds triples to the database based on matching within the existing data (like the WHERE basic graph patterns above); DELETE, as the converse; LOAD, which reads data from a disk file or other source into the graph database; and DROP, which deletes a graph.

The execution flow of a (sub)query is (1) the basic graph pattern in the WHERE, (2) any GROUP BY or HAVING and any aggregation or projection (i.e., the operations after the SELECT keyword), and (3) any trailing solution modifiers, such as ORDER BY or LIMIT.

### 3. ITERATIVE ALGORITHMS IN SPARQL

SPARQL 1.1 as a language does not support iteration, so iterative algorithms will need a construct external to SPARQL to implement iteration. We have used JavaScript and Python scripts to implement the iterative code that calls SPARQL queries. The coarse structure is captured in the following pseudocode, where any of lines 1, 3, 4, or 6 could consist of multiple SPARQL queries.

```

1 establish initial state
2 do {
3   update state
4   measure convergence criteria
5 } while (convergence criteria not met)
6 establish final state and clean up

```

This structure reflects the assumptions that a) the intermediate state updated in line 3 is large enough that we want to retain it within the SPARQL endpoint for performance reasons, rather than transmitting it back to a client system for processing, and b) that the convergence criteria can be summarized to no more than a few scalars (e.g., the number of vertices changing cluster assignment on this iteration, for

peer-pressure clustering). In the algorithmic implementations described here, we have chosen to place intermediate state (*e.g.*, cluster assignment) in a named graph for simplicity (*i.e.*, ease of finding inserted triples for debugging and other purposes) and performance (*i.e.*, eliminating a set of intermediate values via DROP of a named graph is fast compared with finding all the appropriate triples intermingled with other data), but placing the intermediate state in the default graph may be appropriate in some circumstances. Another degree of freedom for the algorithm developer is whether to preserve all intermediate named graphs until the algorithm completes (at the cost of more memory) or to delete intermediate named graphs just after their last use (at the cost of debuggability). Preserving intermediate graphs requires emitting a query where the intermediate-graph name changes from iteration to iteration.

A similar choice for the algorithm developer is whether to place the final results in the default graph or a named graph. Given the extreme flexibility of RDF and SPARQL, an iterative algorithm could be called with different parameters, for example propagating labels through vertices of types A-C on one call and types B-G on a later call; the ability to name a graph with this information simplifies the other URIs within that data (*e.g.*, avoiding the need to express in the predicate (in this example) the specific set of vertex types considered).

A programming note is that nothing in the SPARQL endpoint precludes multiple instances of an iterative algorithm running simultaneously, each sending queries to the endpoint, so some means of avoiding collisions in intermediate graph names will be warranted for general use.

## 4. PEER-PRESSURE CLUSTERING

Peer-pressure clustering belongs to the class of algorithms that are effective by calculating simply on very large data.

### 4.1 The Algorithm

Peer-pressure clustering takes as its input a set of edges, each between a pair from a set of vertices, and calculates each vertex's assignment to a cluster. The number of clusters to be found need not be specified. For heterogeneous graphs, even for clustering vertices of homogeneous type, creating links between the vertices is an initial step whose definition is problem-dependent; see Section 4.6.1 below for details.

Reprising the structure from the figure above, peer-pressure clustering can be expressed via the following pseudocode.

```

1 assign each vertex to an initial cluster
2 do {
3   (re-)assign each vertex to the cluster
      to which a plurality of its
      neighbors belong
4   count the number of vertices that
      changed cluster in the prior step
5 } while (enough vertices changed or other
      criteria)

```

In our implementation the initial assignment is to a cluster with the same name as the vertex.

### 4.2 Relevant Use Cases

Clustering can be useful to understand the group structure of a set of homogeneous vertices. Use cases include the spread of influence in online social networks [8, 2].

## 4.3 Implementation via SPARQL Queries

We chose to place intermediate assignments in named graphs whose names have a common quasi-random seed, "xjz" in the examples below, to avoid collisions.

The first initial query assigns each vertex to a default cluster (named by the vertex name).

```

1 DROP GRAPH <urn:ga/g/xjz0>
2 CREATE GRAPH <urn:ga/g/xjz0>
3 INSERT {
4   GRAPH <urn:ga/g/xjz0>
5     {?s <urn:ga/p/inCluster> ?s }
6 } WHERE {
7   SELECT DISTINCT ?s
8   WHERE {
9     ?s <urn:ga/p/hasLink> ?o .
10  }
11 }

```

Line 1 DROPS (deletes from the database) any existing graph of the same name and thus any triples in such a graph. Line 2 CREATES a new (empty) graph of the same name, which is not strictly necessary before the INSERT but can aid in debugging. The SELECT clause on lines 7-10 finds all vertices in the default graph that are the subject of a `hasLink` predicate and, for each unique such vertex, then on lines 3-5 INSERTs into the named graph a new triple of the same subject, the `inCluster` predicate, and the subject (as the cluster assignment). (We omit the text of a trivial initial query that counts the number of vertices to be clustered.)

The update query works as follows. Lines 13-20, for each vertex, return the vertex, the cluster assignments of its neighbors, and the per-cluster count of neighbor vertices. Lines 10-22 calculate the maximum cardinality of the clusters of the neighbors of each vertex. Lines 23-31 calculate the cluster assignment of that maximum-cardinality cluster. (SPARQL lacks a construct that returns the maximum value of one intermediate result and the corresponding element of another intermediate result.) Lines 8-30 join the maximum cardinality with the cluster name and also, in the case of a tie in maximum cardinality, break any tie by SAMPLEing a cluster assignment for each cluster. Lines 3-6 INSERT the new cluster-assignment triples into the named graph. (For all graph names, the non-SPARQL "[i]" and "[i+1]" syntax denotes that the appropriate iteration count is placed into the string by the code that creates the SPARQL query.)



```

1 DROP GRAPH <urn:ga/g/xjz[i+1]>
2 CREATE GRAPH <urn:ga/g/xjz[i+1]>
3 INSERT
4 {
5   GRAPH <urn:ga/g/xjz[i+1]>
6     { ?s <urn:ga/p/inCluster> ?clus3 }
7 } WHERE {
8   { SELECT ?s (SAMPLE(?clus) AS ?clus3)
9     WHERE {
10      { SELECT ?s
11        (MAX(?clusCt) AS ?maxClusCt)
12      WHERE {
13        SELECT ?s ?clus
14          (COUNT(?clus) AS ?clusCt)
15        WHERE
16          {
17            ?s <urn:ga/p/hasLink> ?o .
18            GRAPH <urn:ga/g/xjz[i]>
19              {?o <urn:ga/p/inCluster> ?clus}
20          } GROUP BY ?s ?clus
21        } GROUP BY ?s
22      }
23    { SELECT ?s ?clus
24      (COUNT(?clus) AS ?clusCt)
25    WHERE
26      {
27        ?s <urn:ga/p/hasLink> ?o .
28        GRAPH <urn:ga/g/xjz[i]>
29          {?o <urn:ga/p/inCluster> ?clus}
30        } GROUP BY ?s ?clus
31      } FILTER (?clusCt = ?maxClusCt)
32    } GROUP BY ?s
33  }
34 }

```

The second query executed in each iteration (below) counts the number of vertices that changed cluster assignment in the just-completed iteration.

```

1 SELECT (COUNT(?oNew) as ?vccCt)
2 WHERE {
3   GRAPH <urn:ga/g/xjzi>
4     {?s <urn:ga/p/inCluster> ?oOld}
5   GRAPH <urn:ga/g/xjzi+1>
6     {?s <urn:ga/p/inCluster> ?oNew}
7   FILTER (?oOld != ?oNew)
8 }

```

The JavaScript code that constructs the queries and calls the SPARQL endpoint is straightforward and hence omitted.

## 4.4 Validation

We initially validated the implementation with synthetic data. The first phase of this was with predictable clustering characteristics, generated with the number of clusters set to  $\log(n)^{1.5}$  where  $n$  is the number of vertices. The generator then considers all edge pairs and adds inter cluster edges with probability  $X$  (0.02 in this case) and intracluster edges with probability  $Y$  (0.1). This data contained 100,000 vertices and 15,736,484 triples.

The second phase of validation with synthetic data was block two-level Erdős-Rényi (BTER) data created by the MATLAB generator by Pinar et al [14], whose output we converted into RDF. The parameters we used, in addition

to power-law degree distribution, were  $\gamma = 2$ ,  $\text{maxdegree} = 100$ ,  $\rho_{\text{init}} = 0.99$ , and  $\rho_{\text{decay}} = 0.8$ . This data contained 1,643,915 vertices and 7,322,102 triples.

## 4.5 Performance

For the initial set of synthetic data (100,000 vertices and 15.7M edges), on a 64-processor, 2TB Urika appliance, peer-pressure clustering converged after 5 iterations, consuming 200.2 seconds in total. For the BTER synthetic data (1.6M vertices and 7.3M edges) it executed for 3h:09m, though it did not converge after 20 iterations, which was the maximum iteration setting. We also tried to apply the algorithm to the Smackdown data created by Mayo Clinic, both small portions and the full 2G (2 billion triples) dataset, where we encountered the quadratic issue described in the following section. We had wondered whether per-query overhead might be a performance issue, but with the overhead far below 1 second, it proved not to be an issue in practice.

## 4.6 Issues Encountered

### 4.6.1 Creating links for clustering

Peer-pressure clustering uses predicates of a given type (`hasLink` in our implementation) as the edges to consider, which can be viewed as similarity links between the vertices. For heterogeneous data, the data must typically be prepared by deciding the similarity criteria, and for vertex pairs which pass the criteria or threshold, creating the edge. We experimented with different similarity functions, mirroring what subject-matter experts may do in practice, tweaking the similarity function until the resulting clusters are useful in the context of the subject matter.

If this approach is used to calculate the edges, the simple approach of comparing all vertices to all other vertices is difficult to scale to large numbers of vertices, as the  $O(n^2)$  cost of this step becomes prohibitively time-consuming for databases containing  $O(100M)$  or more vertices. The clustering algorithm doesn't require all similar vertices to have similarity links, but can work with a more sparsely connected graph, and a sparse pre-processing step would be appropriate for large-scale use. Note that this cost is in the pre-processing step, and that limiting the number of edges created would keep the core peer-pressure clustering algorithm relevant for very large data.

### 4.6.2 SPARQL constructs

Careful readers may note that the inner SELECTs at lines 13-21 and 23-30 in the iterative update query are identical. While developing a complex nested query like this, needing to keep the same code in two spots identical is cumbersome. SPARQL 1.1 contains no good mechanism to define this code once and reuse it, like a function in a procedural language. The SQL WITH clause defines by name such a code block that can be executed wherever its results are needed.

The second and third queries above both have minor changes from one instance to the next (*e.g.*, substituting "xjz2", and "xjz3" into the graph name). While these are not hard to cope with in JavaScript code that creates the queries, it does mean that the query is literally different each time it is executed, and hence the SPARQL endpoint will have to reinterpret and re-optimize the query each time, which could at some point become time-consuming. SQL's *placeholder* capability enables the passing of a value (of a given type) at

execution time that is inserted at the placeholder’s position in the query, avoiding reinterpretation.

## 5. GRAPH DIFFUSION

Graph diffusion is an algorithm that models natural transport phenomena on the connectivity of the graph, much like random walk approaches, but simultaneously moving across all possible edges. Diffusion can be used to characterize semantic data in several ways. It can be used to compute neighborhoods of “close” connectivity or find nodes with similar features. Some applications are models for cascading behaviors such as social network analysis, virus propagation, parallel load balancing, and chemical compound classification [5] [13] [17].

### 5.1 The Algorithm

The graph diffusion algorithm propagates values (typically numeric scores) from specific initial vertices, through connecting edges to neighboring vertices, and by iteration, to the rest of the graph. Each vertex accumulates values as the expanding wavefront propagates through it. Semantic graphs, with named edges, can have a propagation weight assigned to each edge type, increasing or decreasing the diffusion values. In general matrix notation, graph diffusion can be characterized as an iterative update process according to the equation,  $N_i^{t+1} = \sum_j E_{ij} W_{ij} N_j^t$ , where  $N$  is a node (vertex),  $E$  is the binary adjacency matrix (representing edges between nodes), and  $W$  contains the edge weights associated with diffusion. In this formulation,  $E$  and  $W$  are static and can be combined. The diffusion algorithm can run a fixed number of iterations, or (with edge weights < 1.0) can continue until a steady state is reached. There are many algorithm parameters which can be adjusted, including initial conditions, treatment of edges as directed or undirected, and computation of aggregate scores.

A simple implementation of graph diffusion is characterized by this pseudocode.

```

1 assign edge weights by type
2 seed initial diffusion value(s)
3 do {
4     for each vertex with a value {
5         save value as accumulated score
6         propagate value to neighbors
7     }
8 } until completion criteria met

```

### 5.2 Relevant Use Cases

One potential use case for diffusion in healthcare is finding patients with similar clinical features. For example, patients are admitted into a hospital for a variety of different clinical conditions; however, once admitted, apparently dissimilar patients develop common presentations of disease. We applied diffusion to a semantic graph of two years of hospital records for 114,943 patient stays. Diffusion seed values were attached to sub-populations of patients with known conditions of interest. Diffusion values propagated over 74 different edge types (representing demographic, clinical, nursing, and lab measurements) from the initial patients to all others in the dataset. The resulting diffusion values represent each patient’s similarity to the initial patient sub-population.

### 5.3 Implementation via SPARQL Queries

Edge (predicate) weights and diffusion values were stored in named graphs, separating them from each other and the patient data in the default graph. Propagation values were stored in numbered named graphs (*e.g.*, `iter_0`). Propagation values were accumulated for each patient at the end of the process, avoiding per-iteration updates to vertex counters. The SPARQL code for the first diffusion iteration uses values in `iter_0` to create values in named graph `iter_1`.

```

1 PREFIX diff: <urn:diffusion/>
2 DROP GRAPH diff:iter_[i+1];
3 CREATE GRAPH diff:iter_[i+1];
4 INSERT {
5     GRAPH diff:iter_[i+1]
6         {?vertex diff:cntr ?value .}
7 } WHERE {
8     SELECT ?vertex (SUM(?edgeVal) AS ?value)
9     WHERE {
10        GRAPH diff:iter_[i]
11            {?otherVertex diff:cntr ?otherCntr}
12        GRAPH diff:weights
13            {?edge diff:weight ?weight .}
14        { {?otherVertex ?edge ?vertex .}
15          UNION
16            {?vertex ?edge ?otherVertex .} }
17        BIND(?otherCntr*?weight AS ?edgeVal)
18    } GROUP BY ?vertex
19 }

```

The graph pattern in lines 10-11 matches diffusion values from the previous iteration; lines 12-13 identify edge weights in a separate named graph; and the UNION operation at line 15 matches both incoming and outgoing edges of the vertex, making diffusion bidirectional. The BIND on line 17 computes the edge-weighted diffusion value, and the subquery on line 9 aggregates the values. For this implementation, both incoming and outgoing edges of the same type are weighted equally, but a straightforward extension would provide different weights based on edge direction.

All iterations use identical SPARQL code, save for incrementing the graph number on lines 2, 3, 5, and 10. Note that due to the nature of intermediate solution sets and grouping by `?vertex`, this code computes a new diffusion value for a vertex from its neighbors, as opposed to propagating a vertex’s value to its neighbors as shown in the pseudocode.

After diffusion is complete, the per-patient similarity scores are found in the final iteration’s named graph, `iter_4`. More generally, aggregate diffusion scores are computed by summing vertex counters in all iteration-named graphs.

### 5.4 Validation

This implementation of graph diffusion, applied to this in-hospital patient care dataset, generates a score ranking all patients’ similarity to the seed patient. The similarity score is computed over all 74 dimensions (edge types) in the semantic model. Inspection of the values for “similar” and “dissimilar” patients, one dimension at a time, reveals that the most similar patients do, in fact, have characteristics similar to the seed patient.

### 5.5 Performance

We ran four iterations of the diffusion algorithm on a semantic dataset of 89 million vertices and 1.8 billion edges



on a 64 processor Urika appliance with 2 TB of main memory. The first two iterations, propagating from the initial patient URIs to all associated values recorded for those patients, took 202 and 207 seconds, respectively. The third and fourth iterations, propagating back to all other patient URIs, took 1,302 seconds and 1,124 seconds, respectively.

## 5.6 Issues Encountered

For this RDF dataset, extracted from tabular data, the basic structure and relationships between patient URIs, measurement events, and associated data values were well known. This structure allowed us to determine that four iterations of diffusion were sufficient to propagate from the initial patient URI to all other patient URIs. In the more general case, a scoring query would be needed after every UPDATE iteration to determine if the diffusion had completed.

## 6. LABEL PROPAGATION

Label propagation [11] is a clustering algorithm similar to peer-pressure clustering whose purpose is to find clusters of vertices where the clustering is based on the edges linking to neighboring vertices.

### 6.1 The Algorithm

Clustering via label propagation takes as its input a set of edges, each between a pair of vertices, and calculates each vertex's assignment to a cluster based on its neighbors.

Reprising the structure from the figure above, label propagation can be expressed via the following pseudocode:

```

1 assign each vertex to a distinct cluster
2   (cluster label might be some integer
3    primary key of the vertex)
4 do {
5   re-assign each vertex to a cluster based
6   on neighbors in the previous step
7   - choose cluster to which a plurality
8   of first-degree neighbors belong
9   - optional self-voting
10  - tie-breaking rule: sort order of
11  label
12 } while not stopping conditions

```

### 6.2 Implementation via SPARQL Queries

In this implementation, the intermediate assignments are placed in the same (default) graph, but with a distinct predicate name. This makes little difference functionally, and the SPARQL queries are easier to only a minor degree because the syntax for specifying multiple different graphs is more verbose.

The initial cluster label is denoted by the `grouping0` predicate, and assigned the integer primary key of the vertex. The predicate `pkey` is specific to the source data model.

The `p:similar` predicate is also specific to the data model; it represents the network linking the vertices (`?entity`), analogous to `hasLink` in the peer-pressure clustering section.

```

1 INSERT {
2   ?entity lprop:grouping0 ?initial_group
3 } WHERE {
4   ?entity p:similar ?x .
5   ?entity p:pkey ?initial_group .
6 } ;

```

In each iteration, the grouping predicate is incremented to keep them distinct, `lprop:grouping0` in the `WHERE` to look at the previous iteration, and `INSERTing` into `lprop:grouping1`.

The query consists of the `INSERT` and a `WHERE` with nested sub-queries. Sub-queries are joined on bindings of the same names, in the same way as simple *basic graph patterns*. (The query is shown below broken into multiple code listings.)

```

1 INSERT { ?entity lprop:grouping1 ?group .
2 } WHERE {

```

The sub-query selects the entity and group. The `GROUP BY` at the end is on `?entity` because the query needs one solution for each `entity`. The `MIN` of the `group` is chosen to break ties in popularity. That is, if multiple groups have the same popularity, the lowest `pkey` value wins. This is an arbitrary rule, but consistent.

```

1 {
2   SELECT ?entity
3     ( MIN( ?groupx ) AS ?group )
4   WHERE {

```

The first sub-sub-query measures the popularity of the labels of each entity's first-degree neighbors. The `GROUP BY` includes `?entity` to find results per entity; and `?group` so it can `COUNT` the neighbors (`?first`).

In addition, we've chosen to use self-voting, adding 1 to the `popularity` score for the entity's group (`?self`). Note that this query alone might have multiple binding-sets for an entity, one for each group, and that multiple groups might have the same popularity.

```

1 {
2   SELECT ?entity ?groupx
3     (( COUNT( ?first ) +
4      IF( ?self = ?groupx, 1, 0) )
5     AS ?popularity )
6   WHERE {
7     ?entity p:similar ?first .
8     ?entity lprop:grouping0 ?self .
9     ?first lprop:grouping0 ?groupx .
10  } GROUP BY ?entity ?self ?groupx
11 }

```

The next sub-sub-query contains another sub-query which is the same popularity query as above, with the binding names changed so the outer query can use the correct names. The outer query selects the `MAX popularity` for each entity. However, SPARQL does not have a direct mechanism to select the group(s) that have that popularity. This is the reason for duplicating the popularity query. This query below and the one above are joined on `?entity` and `?popularity`, which leaves the `?groupx` in the results.

(Closing out the query is the `GROUP BY` explained above.)

```

1 {
2   SELECT ?entity
3     ( MAX( ?popularityx ) AS ?
4     popularity )
5   WHERE {
6     {
7       SELECT ?entity ?groupx
8         ( ( COUNT( ?first ) +
9          IF( ?self = ?groupx, 1,
10          0 ) )

```

```

9      AS ?popularityx )
10     WHERE {
11       ?entity p:similar ?first .
12       ?entity lprop:grouping0 \\?
13         self .
14       ?first lprop:grouping0 \\?
15         groupx .
16     }
17     GROUP BY ?entity ?self ?
18       groupx
19 } GROUP BY ?entity
20 }
21 }

```

A variation would be to count second-degree (or greater) neighbors instead of (or besides) first-degree neighbors.

There are a number of conditions for stopping the iteration. As with the peer-pressure algorithm above, the percent or number of vertices changed, and simply the maximum number of iterations are considered. There is also a property to the algorithm such that one iteration can make numerous changes, but the next iteration will reverse them, resulting in oscillation that only ends with the max number of iterations. Our implementation detects this by counting the differences between each iteration (below `grouping0` and `grouping1`), then also the previous iteration (which would be `grouping0` and `grouping2`). If either of these `group_diff_counts` are 0, the algorithm halts.

```

1 SELECT ( COUNT( ?entity ) AS ?
2         group_diff_count )
3 WHERE {
4   ?entity lprop:grouping0 ?label .
5   ?entity lprop:grouping1 ?next .
6   FILTER( ?label != ?next )
7 }

```

### 6.3 Validation

We generated synthetic, semi-random data, with some number of groups expected, and some links fully random, not belonging to the expected groups.

When run on real data, reports were generated by SPARQL queries (not shown) to show the averages and modes of various attributes for each group, from which we could see that the groups did have distinct sets of attributes.

### 6.4 Performance

Generation of synthetic data was done in Python, but all other processing of big data was performed by SPARQL within the Urika database. A 64-processor, 2TB Urika appliance was used for running label propagation.

Using a semi-random network of 1.8M vertices and 8M edges, the process stopped after 20 iterations, in 16 minutes, resulting in 48 groups. With 3.6M vertices and 16M edges, the process stopped after 20 iterations, in 26 minutes, resulting in 49 groups.

Calculating the same “popularity” sub-query twice in each iteration is redundant and expensive. The alternative is to first insert the results of that once, then use the results twice.

The trade-off is the cost of inserting a very large result set, which may take more time than building it twice.

## 6.5 Issues Encountered

Long SPARQL queries with repeated (and slightly modified) sections are difficult to maintain. The program driving the iteration and calls to the SPARQL endpoint was implemented in Python. A simple module was written to template SPARQL text to make portions reusable within a query and across queries, while avoiding the need to mix Python and SPARQL code.

Manually debugging any algorithm like this can be challenging, especially with large data. Since the data never leaves the database during processing, the usual practices for debugging software (such as a Python script) do not apply. In addition, the algorithm operates differently from how many scripts are written: breadth first rather than depth first. Using a smaller dataset makes the manual inspection approachable, but this changes the results. Also, to understand the results of each iteration requires more SPARQL queries to inspect or count the results.

Running this type of algorithm can take many hours. If the process stops early because of error, or because the user needs it stopped, starting over from the beginning wastes a lot of time. Inserting metadata into the database itself as each step is completed, allows for the process to be restarted, continuing where it was stopped.

## 7. DISCUSSION

SPARQL as a query language possesses several positive features that lend to rapid prototyping of graph analysis workflows. These features include easy graph preparation, the expressibility of algorithms in the language, and the accessibility of the language to individuals who possess relevant domain expertise but may not possess sufficient knowledge of lower-level graph languages to effectively use them.

Building graphs from real world data on which to apply these algorithms tends to require graph preparation. This graph preparation step is expressible in SPARQL. This step serves to focus analyst and developer attention on questions about which types of data and aspects of that data are most relevant to the domain-specific problem being analyzed. This focusing is an important part of the graph workflow prototyping process as it forces critical thought to the practicalities of the data representation and associated algorithms.

The three algorithms have different execution speeds, measured in edges/second, reflecting the different natures of the algorithms and the newness of these results. In future work, we will explore how much of these speed differences are inherent to the algorithms and how much could be overcome by query changes or query-engine optimizations.

Iterative algorithms are a valuable component of graph analysis workflows. These algorithms produce analytically relevant results, as shown in the rest of the paper, and they are expressible in SPARQL.

Developing, prototyping, and evaluating graph analysis workflows tend to be laborious, human-intensive processes. This development and prototyping requires expertise in graph analysis and the specific domain under analysis. Practitioners with these experiences tend to have less knowledge and familiarity with lower-level graph processing frameworks and languages. SPARQL as a high-level graph query language is

more approachable to these practitioners and requires less time to learn than C++ or another lower-level language. This smaller ramp-up time translates to a faster idea-to-functioning-workflow when using SPARQL than other graph-workflow development languages.

## 8. SUMMARY

We present a method of mapping iterative algorithms on to a combination of SPARQL queries and code in a procedural language (Python and JavaScript, in our examples) that calls the SPARQL queries. We find that writing iterative algorithms in this style is straightforward to implement, with scalability to very large data and good performance. Though there are algorithms for which this approach may be problematic (*e.g.*, when iterative queries are so simple that query overhead is a performance issue, or the intermediate state between iterations is prohibitively large), initially implementing such algorithms in this style delivers correct answers quickly, with an optimized implementation possible via other means if needed.

## 9. ACKNOWLEDGMENTS

The authors from UC Santa Barbara were supported in part by DOE Office of Science contract DE-AC02-05-CH-11231, NSF grant CNS-0709385, a contract from Intel Corp., and a gift from Microsoft Corp.

## 10. REFERENCES

- [1] Apache Jena Project. Java framework for building linked data applications., 2011-2013. <http://jena.apache.org/>.
- [2] M. Cha, A. Mislove, and K. P. Gummadi. A measurement-driven analysis of information propagation in the Flickr social network. In *Proceedings of the 18th International Conference on World Wide Web*, pages 721–730. ACM, 2009.
- [3] Franz, Inc. AllegroGraph ontology modeling capabilities, 2013. <http://www.franz.com/agraph/>.
- [4] P. Gearson, A. Passant, and A. Polleres. SPARQL 1.1 update (W3C recommendation 21 march 2013). Technical report, World Wide Web Consortium, 2013. <http://www.w3.org/TR/sparql11-update/>.
- [5] D. Gruhl, R. Guha, D. Liben-nowell, and A. Tomkins. Information diffusion through blogspace. In *In WWW '04*, pages 491–501. ACM Press, 2004.
- [6] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2-3), 2005.
- [7] S. Harris and A. Seaborne. SPARQL 1.1 query language (W3C recommendation 21 march 2013). Technical report, World Wide Web Consortium, 2013. <http://www.w3.org/TR/sparql11-query/>.
- [8] P. Hui and S. Buchegger. Groupthink and peer pressure: Social influence in online social network groups. In *Proceedings of International Conference on Advances in Social Network Analysis and Mining (ASONAM)*. IEEE, 2009.
- [9] Network Working Group. Uniform resource identifier (URI): Generic syntax (RFC 3986). Technical report, Internet Engineering Task Force, 2005. <http://tools.ietf.org/html/rfc3986>.
- [10] OpenRDF Project Team. OpenRDF.org ... home of sesame, 1997-2012. <http://www.openrdf.org/>.
- [11] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3), March 2007.
- [12] RDF Working Group. Resource Description Framework (RDF). Technical report, World Wide Web Consortium, 2004. <http://www.w3.org/RDF/>.
- [13] S. Schamberger. On partitioning fem graphs using diffusion. In *Proceedings of 18th International Parallel and Distributed Processing Symposium*, 2004.
- [14] C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E*, 85(5), May 2012.
- [15] V. B. Shah. *An Interactive System for Combinatorial Scientific Computing with an Emphasis on Programmer Productivity*. PhD thesis, University of California, Santa Barbara, 2007.
- [16] A. Shkapsky, K. Zeng, and C. Zaniolo. Graph queries in a next-generation datalog system. In *Proceedings of IEEE Conference on Very Large Databases*. IEEE, 2013.
- [17] A. Smalter, J. Huan, Y. Jia, and G. Lushington. Gpd: A graph pattern diffusion kernel for accurate graph classification with applications in cheminformatics. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 7(2):197–207, 2010.
- [18] TopQuadrant. TopBraid Composer modeling environment, 2013. [http://www.topquadrant.com/products/TB\\_Composer.html](http://www.topquadrant.com/products/TB_Composer.html).
- [19] YarcData LLC, a Cray Company. Urika graph-analytic appliance, 2013. <http://yarcdata.com/Products/>.

# A Map-Reduce algorithm for querying linked data based on query decomposition into stars\*

Christos Nomikos

Department of Computer Science and Engineering  
University of Ioannina, Ioannina, Greece  
cnomikos@cs.uoi.gr

Manolis Gergatsoulis, Eleftherios Kalogeros, Matthew Damigos

Database & Information Systems Group (DBIS)  
Department of Archives, Library Science and Museology  
School of Information Science and Informatics  
Ionian University, Corfu, Greece  
manolis@ionio.gr

## ABSTRACT

In this paper, we investigate the problem of efficient querying large amount of linked data using Map-Reduce framework. We assume data graphs that are arbitrarily partitioned in the distributed file system. Our technique focuses on the decomposition of the query posed by the user, which is given in the form of a query graph into star subqueries. We propose a two-phase, scalable Map-Reduce algorithm that efficiently results the answer of the initial query by computing and appropriately combining the subquery answers.

## Categories and Subject Descriptors

H.2 [Database Management]: Systems—*Query Processing*; C.2.4 [Distributed Systems]: Distributed Databases; D.1.3 [Concurrent Programming]: Distributed programming

## General Terms

Algorithms, Theory, Experimentation

## Keywords

Linked Data, Graph Querying, Map-Reduce, Distributed Processing, Cloud Computing, Semantic Web

---

\*This research was supported by the project “Handling Uncertainty in Data Intensive Applications”, co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program “Education and Lifelong Learning”, under the research funding program THALES.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

## 1. INTRODUCTION

Linked data is a widespread method for publishing interlinked data, built upon a standard model used for data interchange, called RDF. As the amount of linked data is rapidly increasing the efficient management, analysis, and hence querying of large amount of linked data has become a significant challenge in many areas, such as learning analytics, digital libraries, and other applications analyzing linked open data. The efficient querying of large amount of data is also a significant challenge in many other information management areas, where parallel processing has been proved particularly effective in manipulating such an amount of data.

A well-established programming framework used for processing and managing large amount of data, in parallel, using a cluster of commodity machines is Map-Reduce [8]. A popular, open-source implementation is Apache Hadoop [1]. Boasting a simple, fault-tolerant and scalable paradigm, Hadoop has been established as dominant in the area of massive data analysis.

In this paper, we investigate the problem of efficient querying large amount of linked data using Map-Reduce framework, and extend our approach presented in [10]. In particular, we focus on the decomposition of the query posed by the user, which is given in the form of a query graph, into star subqueries and propose a two-phase, scalable Map-Reduce algorithm that efficiently results the answer of the initial query. Furthermore, we assume data graphs that are arbitrarily partitioned in the distributed file system. The first phase of our algorithm takes advantage of the star form of the sub-queries and focuses on evaluating the star subqueries over the input segments. The results of the sub-queries are emitted to the second phase, which combines them properly in order to produce the answers of the initial query.

## 2. RELATED WORK

During the last decade, the problem of efficiently querying large data graphs using Map-Reduce framework has been investigated in many research areas related to information management [9, 2, 12, 11, 16, 15, 13, 18, 5, 6, 19].

Evaluating SPARQL queries over RDF graphs by parallelizing the processing of the join and selection operators

has received much attention [14, 12, 16, 15, 18]. The RDF triples can be either directly stored in files in the distributed file system (DFS) or stored in a distributed database (e.g., HBase [16]). In [14], the authors propose an approach where the query plan is evaluated using a sequence of Map-Reduce phases. Initially, the relevant RDF triples are selected and then a sequence of joins is evaluated. Each operator in the plan is performed within a Map-Reduce phase, while the evaluation of the query requires multiple Map-Reduce phases, according to the number of operators of the given query. In [12] the RDF triples are stored in multiple DFS files, according to their predicates and objects, while only the relevant files are read from the DFS for each query. In H<sub>2</sub>RDF system [16] data triples are stored in HBase. In order to answer a query, a sequence of joins is executed, which is obtained in a greedy manner. Other approaches have been proposed, which translate SPARQL queries to other query languages, such as JACL ([15]) and PigLatin ([13], [18]). In addition, [7] provides a detailed experimental analysis and comparison of the main NoSQL data stores for RDF processing. The systems that are used for storing the data are the following: Apache HBase, CumulusDB (a RDF data store built upon Cassandra) and Couchbase; while the 4store was used as a baseline, native and distributed RDF DBMS. Jena was used as a SPARQL query engine over HBase and Couchbase, Hive over HBase and Sesame for CumulusDB.

In [11] and [9], the RDF data has been initially partitioned in a predefined manner. [11] assumes that the RDF graph is vertex partitioned and its parts are stored with some triple replication, to ensure that for small queries, each answer can be computed using a single part of the graph. Larger queries are decomposed and the answers to the subqueries are joined by MapReduce jobs. In HadoopRDF [9], RDF triples with the same predicate are placed in the same part of the data graph, which is stored in a traditional triple store, such as Sesame. Queries are divided in the same way, so that each subquery can be answered by a single computer-node. The answers to the subqueries are merged by MapReduce jobs. The decomposition of the given query is also one of the main characteristics of our Map-Reduce algorithm presented in [10]. In that approach, however, the triples could be arbitrarily partitioned in the DFS and the queries are decomposed into paths. The proposed Map-Reduce algorithm evaluates the given query in two phases; one for evaluating the path subqueries and one for finding the total answers by combining the results of the subqueries.

The problem of querying large data graphs using Map-Reduce is also investigated in [2, 19, 6]. J. Cohen in [6] presents different approaches of finding subgraphs in large data graphs using Map-Reduce. In [19], the authors focus on the problem of querying triangles and propose one single-phase and one two-phase Map-Reduce algorithm for finding triangles in a graph whose edges have been arbitrarily distributed in DFS. Afrati et al. [2] investigate the cost of evaluating query graphs on data graphs using Map-Reduce, and proposed an approach of translating the graphs into conjunctive queries which in turn are evaluated in Map-Reduce using the approach proposed in [3].

Systems for querying RDF data that are distributed over the web, which adopt query decomposition, have also been proposed ([17], [4]). DARK [17] uses a service description language to maintain information about the data stored in various hosts, and uses these service descriptions for query

planning and optimization. Avalanche [4] has a preprocessing phase that selects a set of candidate host, which are queried for statistical information. Next, the query execution phase breaks down the given query into subqueries (called molecules), which are bound to physical hosts by a plan generator. This phase terminates, when an adequate number of solutions have been found.

### 3. DATA AND QUERY GRAPHS

In this section we introduce the basic notions regarding our data model. We start with the definition of data and query graphs:

*Definition 1.* Let  $U_{so}$  and  $U_p$  be two disjoint infinite sets of URI references and let  $L$  be an infinite set of (plain) literals<sup>1</sup>. An element  $(s, p, o) \in U_{so} \times U_p \times (U_{so} \cup L)$  is called a *data triple*. In a data triple  $(s, p, o)$ ,  $s$  is called the *subject*,  $p$  the *predicate* and  $o$  the *object*. A *data graph* is a non-empty set of data triples. A data graph  $G'$  is a *subgraph* of a data graph  $G$  if  $G' \subseteq G$ .

*Definition 2.* Let  $U_{so}$  and  $U_p$  be two disjoint sets of URI references, let  $L$  be a set of (plain) literals and let  $V$  be a set of variables. An element  $(s, p, o) \in (U_{so} \cup V) \times U_p \times (U_{so} \cup L \cup V)$  is called a *query triple*. A *query graph* (or simply a *query*) is a non-empty set of query triples. The *output pattern*  $O(Q)$  of a query graph  $Q$  is the sequence  $(X_1, \dots, X_n)$  of the variables appearing in  $Q$ . A query graph  $Q'$  is a *subquery* of a query graph  $Q$  if  $Q' \subseteq Q$ .

Notice that, queries with variables in the place of predicates are not allowed. The set of *nodes*  $nd(G)$  of a data graph  $G$  consists of all the elements of  $U_{so} \cup L$  that occur in the triples of  $G$ . The set of *arc labels*  $al(G)$  of a data graph  $G$  consist of all the elements of  $U_p$  that occur in the triples of  $G$ . The set of nodes  $nd(Q)$  and the set of arc labels  $al(Q)$  of a query  $Q$  are defined in an analogous way.

A class of queries of a special form (namely star queries) play an important role in this paper.

*Definition 3.* A query  $Q$  is called a *star query* if there exists a node  $c \in nd(Q)$  such that for every triple  $t \in Q$  it is either  $t = (c, p, v)$  or  $t = (v, p, c)$  for some node  $v \in nd(Q)$  and some predicate  $p \in al(Q)$ . The node  $c$  is called the *central node* of  $Q$ .

It is convenient to use a graphical representation for data and query graphs. A node (subject or object) which is either a URI or a variable is represented as a rounded rectangle while an object which is a literal is represented by a rectangle. A triple  $(s, p, o)$  is represented by an arc from  $s$  to  $o$ , labeled with  $p$ . Moreover, we adopt the following conventions: strings with initial lowercase letters represent predicates, while strings with initial uppercase letters denote URIs. Literals are represented as strings enclosed in double quotes. Finally, variable names begin with the question mark symbol (?).

*Example 1.* Fig. 1 depicts a data graph  $G$  and a query graph  $Q$ . □

<sup>1</sup>In this paper we do not consider typed literals

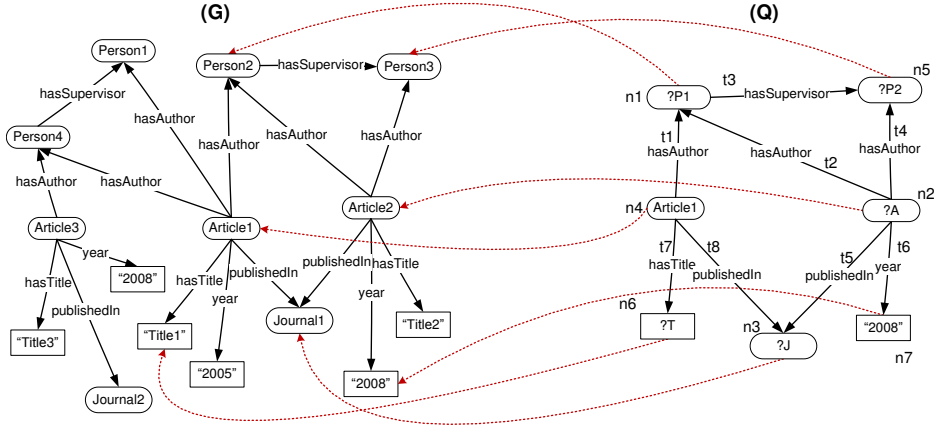


Figure 1: An embedding of the query graph  $Q$  in the data graph  $G$ .

In order to compute the answers to a query  $Q$  for a given data graph  $G$ , we need to find an appropriate correspondence between the nodes of  $Q$  and the nodes of  $G$ . This is formalized by the notion of embedding, defined as follows:

*Definition 4.* An *embedding* of a query graph  $Q$  in a data graph  $G$  is a total mapping  $e : nd(Q) \rightarrow nd(G)$  with the following properties:

1. For each non-variable node  $v \in nd(Q)$ , it is  $e(v) = v$ .
2. For each triple  $(s, p, o) \in Q$ ,  $(e(s), p, e(o))$  is in  $G$ .

The tuple  $(e(X_1), \dots, e(X_n))$ , where  $(X_1, \dots, X_n)$  is the output pattern of  $Q$ , is said to be an *answer* to the query  $Q$ .

*Example 2.* Fig. 1 shows an embedding of the query  $Q$  in data graph  $G$ . The answer obtained is  $(?P1, ?A, ?J, ?P2, ?T) = (Person2, Article2, Journal1, Person3, "Title1")$ .  $\square$

## 4. DATA GRAPH PARTITIONING

Data graphs may consist of a huge number of data triples, stored in numerous computer nodes. In this section we define the notion of the partition of a data graph.

*Definition 5.* A *triple partition* of a data graph  $G$  is a tuple  $\mathcal{P} = (G_1, \dots, G_m)$  where  $G_1, \dots, G_m \subseteq G$ , such that  $\bigcup_i G_i = G$  and  $G_i \cap G_j = \emptyset$ , for all  $i, j$ , with  $1 \leq i < j \leq m$ . Subgraphs  $G_1, \dots, G_m$  are called the *graph segments*.

From the above definition it follows that graph segments in a triple partition of a graph  $G$  cannot share data triples; however, they may have common nodes.

*Definition 6.* Let  $\mathcal{P} = (G_1, \dots, G_m)$  be a triple partition of a data graph  $G$ . Then, a *border node*  $v$  of  $G_i$ , is a node that belongs to  $nd(G_i) \cap nd(G_j) \cap U_{so}$  for some  $j \neq i$ . We denote by  $B(G_i)$  the set of border nodes of  $G_i$ .

*Example 3.* (Continued from Example 2). In Fig. 2 we see a triple partition of the data graph  $G$  of Fig. 1. The shaded nodes correspond to the border nodes between the graph segments. Consider now the query graph  $Q$  appearing in the right part of Fig. 2. It is easy to see that we cannot obtain the solution appearing in Example 2 by finding an embedding of  $Q$  in a segment of  $G$  appearing in Fig. 2 (as such an embedding does not exist).  $\square$

## 5. QUERY DECOMPOSITION

In this section we define the notion of query decomposition and we show how it can be used in order to compute all the answers to a given query.

*Definition 7.* A *query decomposition* of a query graph  $Q$  is a tuple  $\mathcal{F} = (Q_1, \dots, Q_n)$  such that  $Q_1, \dots, Q_n \subseteq Q$  and  $\bigcup_i Q_i = Q$ . A query decomposition is *non-redundant* if  $Q_i \cap Q_j = \emptyset$  for each pair  $i, j$  such that  $1 \leq i < j \leq n$ . A *branching node* in  $Q$  is a node that belongs to  $nd(Q_i) \cap nd(Q_j)$  for a pair  $i, j$ , where  $i \neq j$ . By  $B(Q)$  we denote all branching nodes of  $Q$ .

In this paper, our aim is to construct the embeddings of a query  $Q$  in  $G$ , by appropriately combining embeddings of its subqueries in  $G$ . Two embeddings can be combined only in the case that they agree in the values of nodes that are common in the corresponding subqueries. The above requirement is formalized by the following definition:

*Definition 8.* Let  $Q_1, Q_2$  be two query graphs and let  $e_1, e_2$  be embeddings of  $Q_1, Q_2$  respectively in a data graph  $G$ . We say that  $e_1$  and  $e_2$  are *compatible* if for every  $v \in nd(Q_1) \cap nd(Q_2)$  it is  $e_1(v) = e_2(v)$ . The *join* of  $e_1$  and  $e_2$  is the embedding  $e$  of  $Q_1 \cup Q_2$  in  $G$  defined as follows:

$$e(v) = \begin{cases} e_1(v) & \text{if } v \in nd(Q_1) \\ e_2(v) & \text{otherwise} \end{cases}$$

It is not hard to see that the join operation is commutative and associative. Therefore, we can refer to the embedding resulting by the join of  $n$  mutually compatible embeddings without ambiguity. The following theorem can be easily proved by an induction on the number  $n$  of subqueries in the decomposition of a query  $Q$ .

**THEOREM 1.** *Let  $\mathcal{F} = (Q_1, \dots, Q_n)$  be a query decomposition of a query graph  $Q$  and let  $G$  be a data graph. Then  $e$  is an embedding of  $Q$  in  $G$  if and only if there exist mutually compatible embeddings  $e_1, \dots, e_n$  of  $Q_1, \dots, Q_n$  in  $G$  such that the join of  $e_1, \dots, e_n$  is  $e$ .*

The above theorem implies that in order to compute the answers to a given query for a data graph  $G$ , we can decompose the query into subqueries that belong to a certain class

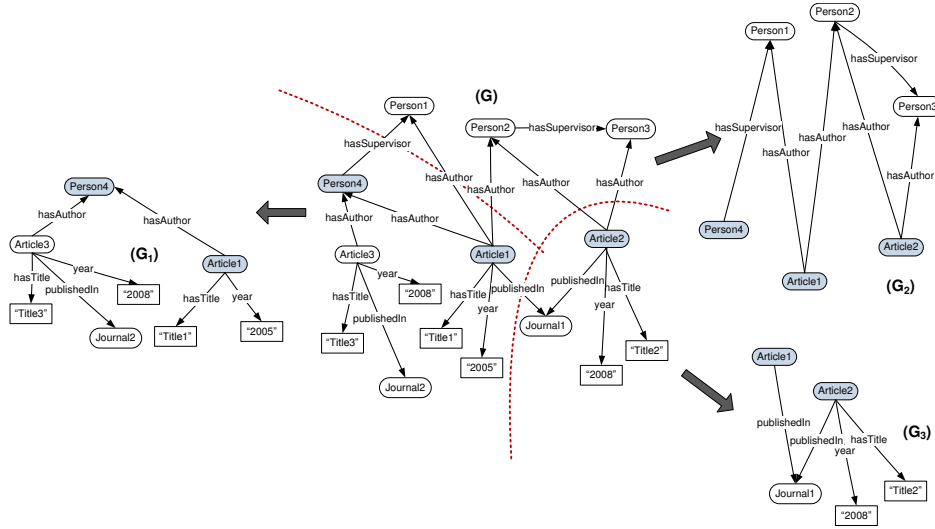


Figure 2: triple partition of the data graph  $G$  of Fig. 1.

$C$ , compute the embeddings of the subqueries in  $G$  (which may be an easier task due to the special form of the subqueries) and then join these embedding to obtain the desired result. However, given a target class of queries  $C$ , it may not be always possible to decompose an arbitrary query  $Q$  into subqueries that belong to  $C$  (for example this is the case if  $C$  is the class of path queries of length 3). Nevertheless, for every query there exist a non-redundant decomposition into star subqueries. This follows trivially from the fact that every query that consists of a single triple is a star query (with either the subject or the object being the central node). We next present a more general result, relating the decompositions of a query graph into star subqueries to the node covers of this query graph.

*Definition 9.* Let  $Q$  be a query graph. A set of nodes  $V \subseteq nd(Q)$  is called a *node cover* of  $Q$  if for every triple  $(s, p, o) \in Q$ , it is either  $s \in V$  or  $o \in V$ . A node cover  $V$  is *minimal* if no proper subset of  $V$  is a node cover.

**LEMMA 1.** Let  $Q$  be a query graph and let  $V = \{v_1, \dots, v_k\}$  be a minimal node cover of  $Q$ . For each  $v_i$  define the star query  $Q_{v_i} = \{t \in Q \mid t = (s, p, v_i)\} \cup \{t \in Q \mid t = (v_i, p, o) \text{ and } o \notin V\}$ . Then  $\mathcal{F} = (Q_{v_1}, \dots, Q_{v_k})$  is a non-redundant decomposition of  $Q$ .

Therefore, if a set of nodes is a minimal node cover of a query  $Q$ , then its elements are the central nodes of the star subqueries in some non-redundant decomposition of  $Q$ .

Conversely, in any decomposition (redundant or not) of a query into stars, the set of the central nodes is a node cover (not necessarily minimal).

**LEMMA 2.** Let  $Q$  be a query graph, let  $\mathcal{F} = (Q_1, \dots, Q_k)$  be a decomposition of  $Q$  such that  $Q_1, \dots, Q_k$  are star queries and let  $c_1, \dots, c_k$  be their central nodes. Then  $\{c_1, \dots, c_k\}$  is a node cover of  $Q$ .

*Example 4.* In Fig. 3 we see a decomposition of the query  $Q$  into three star queries  $Q_1, Q_2$  and  $Q_3$ , which is obtained by the construction of Lemma 1, using the minimal node cover  $\{n_4, n_2, n_5\}$  of  $Q$ .  $\square$

To summarize, suppose that a data graph  $G$  is partitioned into  $m$  segments  $G_1, \dots, G_m$ , that are stored in different computer nodes. The above discussion suggests the following query evaluation strategy:

- Step 1:* Decompose query  $Q$  into star subqueries  $Q_1, \dots, Q_n$ .
- Step 2:* Compute all possible embeddings of each triple in  $Q$  in every segment  $G_i$  of  $G$ .
- Step 3:* For each subquery  $Q_j$ , collect the embeddings of all triples in  $Q_j$  and join compatible embeddings in all possible ways to compute the embeddings of  $Q_j$  in  $G$ .
- Step 4:* Join compatible embeddings  $Q_1, \dots, Q_n$  in all possible ways to compute the embeddings of  $Q$  in  $G$ .

## 6. A MAP-REDUCE ALGORITHM

We start this section with a brief presentation of the Map-Reduce framework. Then, we give a detailed description of our algorithm for querying linked-data using Map-Reduce.

### 6.1 The MapReduce framework

Map-Reduce is a programming framework for processing large datasets in a distributed manner, using a cluster of commodity machines. The storage layer for the Map-Reduce framework is a Distributed File System (DFS), such as the Hadoop Distributed File System (HDFS). The DFSs differ from conventional file systems in three main aspects. First, the data files are distributed across the nodes of the cluster. Second, their block/chunk size (typically 16-128MB in most of DFSs) is much larger than those in conventional file systems. Third, replication of chunks in relatively independent locations ensures availability.

The framework is based on the definition of two functions, the *Map* and the *Reduce* function. In particular, the user defines the two functions, which run in each cluster node, in isolation. The map function is applied to one or more files, in DFS, and results **[key, value]** pairs. This process is called *Map process/task*. The nodes that run the Map processes are called *Mappers*, and may run multiple tasks over different input files. The *master controller* is responsible to route the pairs to the *Reducers* (i.e., the nodes that apply the reduce function to the pairs) such that all pairs with the same key



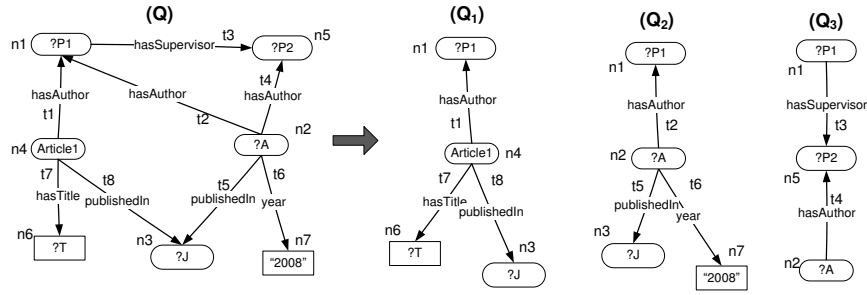


Figure 3: Query decomposition.

initialize a single reduce process, called *reduce task*. The reduce tasks apply the reduce function to the input pairs and also results [key, value] pairs, which are eventually stored in the DFS. This procedure describes one *MapReduce phase*. Furthermore, the output of the reducer can be set as the input of a map function, which gives the user the flexibility to create workflows consisting of Map-Reduce phases.

## 6.2 The preprocessing phase

In this phase query  $Q$  is decomposed into a set of star subqueries  $Q_1, \dots, Q_n$ . For each subquery  $Q_i$ , a *query prototype* of the form  $(bnFlags, nbnFlags, tFlags)$  is constructed, where  $bnFlags$ ,  $nbnFlags$  and  $tFlags$  have one place for each branching node, non-branching node, and triple in  $Q$ , respectively. If an element (node or triple) occurs in  $Q_i$ , then the corresponding place in the query prototype contains a “+”, otherwise it contains a “-”. Moreover, an auxiliary list  $NBL = [(b_i, Q_j) \mid b_i \in B(Q) \text{ and } b_i \notin nd(Q_j)]$  is constructed.

Preprocessing phase emits the above to the mappers of Phase 1 with key the pair  $(subqueryID, SegmentID)$ .  $NBL$  is also emitted to all reducers of Phase 1.

For the presentation of the algorithm we assume an enumeration  $n_1, n_2, \dots, n_{|nd(Q)|}$  of the nodes of a query  $Q$ , such that  $n_1, n_2, \dots, n_{|B(Q)|}$  are the branching nodes of  $Q$  and  $n_{|B(Q)|+1}, \dots, n_{|nd(Q)|}$  are the non-branching nodes of  $Q$ . We will denote by  $I$  the function that gives the index of a node in  $nd(Q)$  with respect to the above enumeration (that is, for every  $x \in nd(Q)$  it holds  $x = n_I(x)$ ). We also denote by  $I_{nb}$  the function from  $nd(Q) - B(Q)$  to  $\{1, \dots, |nd(Q) - B(Q)|\}$ , with  $I_{nb}(x) = I(x) - |B(Q)|$ . Similarly, we assume an enumeration  $t_1, t_2, \dots, t_{|Q|}$  of the triples in  $Q$ .

An embedding  $e$  of a (sub)query is represented as a pair of tuples  $(bn, nbn)$ . If  $x$  is a branching (resp. non-branching) node, then  $bn[I(x)]$  (resp.  $nbn[I_{nb}(x)]$ ) contains the value  $e(x)$ . If  $e$  is an embedding of a subquery  $Q_i$  and  $x$  is a node that does not occur in  $Q_i$ , then an asterisk (“\*”) is stored in the place of  $e(x)$ .

*Example 5.* Consider the query graph appearing in Fig. 3 decomposed into three star subqueries. The branching nodes are  $n_1, n_2$  and  $n_3$ , while the non-branching nodes are  $n_4, n_5, n_6$  and  $n_7$ . The query prototypes are the following:

$Q_1: ((+, -, +), (+, -, -, +), (+, -, -, -, +, +))$

$Q_2: ((+, +, +), (-, -, -, +), (-, +, -, -, +, -, -))$

$Q_3: ((+, +, -), (-, +, -, -), (-, -, +, +, -, -, -))$

The list  $NBL = [(n_2, Q_1), (n_3, Q_3)]$  is also constructed.  $\square$

## 6.3 Phase 1 of the algorithm

The first phase of the algorithm computes the embeddings of the star subqueries  $Q_1, \dots, Q_n$  in  $G$ .

### 6.3.1 Mapper of Phase 1

Each mapper gets as input a graph segment  $G_j$ , a star subquery  $Q_i$  and the  $NBL$  list. We denote by  $c_i$  the central node of  $Q_i$  (recall that this node appears in every triple of  $Q_i$ ). The operation of the mapper is divided into two parts. Observe that if for some embedding  $e$  of  $Q_i$  in  $G$  the value of  $c_i$  is a non-border node of  $G_j$  (i.e., is  $e(c_i) \in (nd(G_i) - B(G_i))$ ), then it holds  $e(v) \in G_j$  for every node  $v \in nd(Q_i)$ . This means that  $e$  is an embedding of  $Q_i$  into  $G_j$  and it can be computed locally. This computation is performed by the second part of the mapper.

The first part of the mapper handles the information that is relevant to the remaining embeddings: it computes all the embeddings of each triple of  $Q_i$  in  $G_j$  that map the central node  $c_i$  to a border node, and emits the results to appropriate reducers. More specifically, let  $t = (s, p, o)$  be a triple that belongs to subquery  $Q_i$  and let  $e$  be an embedding of  $t$  into  $G_j$  such that  $e(c_i)$  is a border node of  $G_j$ . If the central node of  $Q_i$  is  $s$  then the mapper emits a pair  $(key, value)$ , where  $key = (Q_i, e(s))$  and  $value = (o, e(o))$ . Otherwise (i.e., the central node of  $Q_i$  is  $o$ ) then  $key = (Q_i, e(o))$  and  $value = (s, e(s))$ .

Notice that embeddings of triples in  $Q_i$  that map  $c_i$  to different nodes of the data graph are incompatible and cannot be joined to obtain an embedding of  $Q_i$  into  $G$ . Since the value of  $c_i$  is included in the key, incompatible embeddings of triples are emitted to different reducers, while compatible embeddings are emitted to the same reducer.

The second part of the mapper, computes all the embeddings of  $Q_i$  in  $G_j$ , which map  $c_i$  to a non-border node of  $G_j$ . This can be achieved either by adding an appropriate conjunct to  $Q_i$ , or by computing all the embeddings of  $Q_i$  in  $G_j$  and then removing those that assign border nodes to  $c_i$ . The embeddings computed in the second part of the mapper are directly emitted to the mappers of Phase 2 (rather than to the reducers of Phase 1). Similarly, the values of branching nodes are emitted to the mappers of Phase 2.

```

mapper1((Q_i, G_j), (G_jData, B(G_jData), subqueryInfo, NBL))
// (Q_i, G_j): Q_i/G_j is the ID of a subquery/data segment
// G_jData: the content of the data graph segment G_j
// B(G_jData): the set of border nodes of G_j
// SubqueryInfo: prototypes/branching & non-branching nodes
// triples of Q
// NBL: the list of missing branching nodes
begin
- c_i := the central node of Q_i
% Part 1
- for each triple t = (c_i, p, o) in Q_i do
begin
- compute E = {e | e is an embedding of {t} in G_jData

```

```

    and  $e(c_i) \in B(G_jData)$  };
    - for each embedding  $e$  in  $E$  do
        emit( $([Q_i, e(c_i)], (o, e(o)))$ )
    end
- for each triple  $t = (s, p, c_i)$  in  $Q_i$  do
    begin
        - compute  $E = \{e \mid e \text{ is an embedding of } \{t\} \text{ in } G_jData \text{ and } e(c_i) \in B(G_jData)\}$ ;
        - for each embedding  $e$  in  $E$  do
            emit( $([Q_i, e(c_i)], (s, e(s)))$ )
        end
    % Part 2
    - compute  $E = \{e \mid e \text{ is a embedding of } Q_i \text{ in } G_jData \text{ and } e(c_i) \notin B(G_jData)\}$ 
    - for each embedding  $e = (bn, nbn)$  in  $E$  do
        begin
            - emitToSecondPhase( $[Q_i, (bn, nbn)]$ );
            - for  $k = 1$  to  $|bn|$  do
                - if  $(bn[k] \neq '*')$  then
                    - for each  $(n_k, Q_j)$  in  $NBL$  do
                        - emitToSecondPhase( $[Q_j, (n_k, bn[k])]$ );
                    end
                end
            end
        end
    end.

```

*Example 6.* (Continued from Example 5). In this example we see the application of the *mapper1* on the pairs of subqueries and graph segments:

Applying *mapper1* on  $(Q_1, G_1)$  results in emission (see Part 1) of the following *key, value* pairs to the *reducer1*:

$(Q_1, Article1), (n_1, Person4)$  (embedding of  $t1$ )  
 $(Q_1, Article1), (n_6, "Title1")$  (embedding of  $t7$ )

No key value pairs are emitted to Phase 2.

Applying *mapper1* on  $(Q_2, G_1)$  results in emission (see Part 1) of the following *key, value* pairs to the *reducer1*:

$(Q_2, Article1), (n_1, Person4)$  (embedding of  $t2$ )

Besides, the following *key, value* pairs are emitted directly (see Part 2) to the *mapper2* (mapper of Phase 2):

$Q_2, (\langle Person4, Article3, Journal2 \rangle, (*, *, *, "2008"))$   
 $Q_1, (n_2, Article3)$   
 $Q_3, (n_3, Journal2)$

Applying *mapper1* on  $(Q_3, G_1)$  results in emission (see Part 1) of the following *key, value* pairs to the *reducer1*:

$(Q_3, Person4), (n_2, Article1)$  (embedding of  $t4$ )

$(Q_3, Person4), (n_2, Article3)$  (embedding of  $t4$ )

No key value pairs are emitted to Phase 2.

Applying *mapper1* on  $(Q_1, G_2)$  results in emission (see Part 1) of the following *key, value* pairs to the *reducer1*:

$(Q_1, Article1), (n_1, Person1)$  (embedding of  $t1$ )

$(Q_1, Article1), (n_1, Person2)$  (embedding of  $t1$ )

No key value pairs are emitted to Phase 2.

Applying *mapper1* on  $(Q_2, G_2)$  results in emission (see Part 1) of the following *key, value* pairs to the *reducer1*:

$(Q_2, Article1), (n_1, Person1)$  (embedding of  $t2$ )

$(Q_2, Article1), (n_1, Person2)$  (embedding of  $t2$ )

$(Q_2, Article2), (n_1, Person2)$  (embedding of  $t2$ )

$(Q_2, Article2), (n_1, Person3)$  (embedding of  $t2$ )

No key value pairs are emitted to Phase 2.

Applying *mapper1* on  $(Q_3, G_2)$  results in no emission of any *key, value* pair to the *reducer1*. However, the following *key, value* pairs are emitted (see Part 2) to the *mapper2*:

$Q_3, (\langle Person4, Article1, * \rangle, (*, Person1, *, *))$

$Q_3, (\langle Person2, Article2, * \rangle, (*, Person3, *, *))$

$Q_1, (n_2, Article1)$

$Q_1, (n_2, Article2)$

Applying *mapper1* on  $(Q_1, G_3)$  results in emission of the following *key, value* pair to the *reducer1*:

$(Q_1, Article1), (n_3, Journal1)$  ( $t8$ )

No key value pairs are emitted to Phase 2.

Applying *mapper1* on  $(Q_2, G_3)$  results in emission of the following *key, value* pair to the *reducer1*:

$(Q_2, Article1), (n_3, Journal1)$  ( $t5$ )

$(Q_2, Article2), (n_3, Journal1)$  ( $t5$ )

$(Q_2, Article2), (n_7, "2008")$  ( $t6$ )

No key value pairs are emitted to Phase 2.

Applying *mapper1* on  $(Q_3, G_3)$  produces no results.  $\square$

### 6.3.2 Reducer of Phase 1

For each key  $(Q_i, v)$  the corresponding reducer computes all the embeddings of  $Q_i$  that map central node  $c_i$  of  $Q_i$  to  $v$ . The input to this reducer is a list of pairs of the form  $(n_k, u)$ , where  $n_k$  is a node of  $Q_i$  different from  $c_i$  and  $u$  is a possible values for  $n_k$  in an embedding of  $Q_i$  in  $G$ .

Suppose that  $n_{k_1}, n_{k_2}, \dots, n_{k_m}$  are the non-central nodes in  $Q_i$ . The reducer constructs for every  $j = 1, \dots, m$  a set  $L_{k_j}$  of all possible values for node  $n_{k_j}$ . Then for each element  $(x_1, x_2, \dots, x_m)$  of the cartesian product  $L_{k_1} \times L_{k_2} \times \dots \times L_{k_m}$  it constructs an embedding  $e = (bn, nbn)$  of  $Q_i$  in  $G$ , such that  $e(c_i) = v$  and  $e(n_{k_j}) = x_j$  and emits  $(Q_i, (bn, nbn))$  (see Subsection 6.2 for the representation of an embedding).

Moreover, if every list  $L_{k_1}, L_{k_2}, \dots, L_{k_m}$  is non-empty (that is, at least one embedding of  $Q_i$  has been found), *reducer1* emits the values of missing branching nodes.

*reducer1*( $(Q_i, v), values$ )

//  $Q_i$ : a subquery ID

//  $v$ : the value of the central node of  $Q_i$

//  $values$ : contains a list of pairs  $(x, u)$  and the  $NBL$  list.

**begin**

- *allNonEmpty* := true

- for each non-central node  $x$  in  $Q_i$  do

begin

-  $L[x] := \{u \mid (x, u) \in values\}$

- if  $L[x]$  is empty then *allNonEmpty* := false

end

- if *allNonEmpty* then

begin

- create an embedding with undefined values

$(bn, nbn) := ((*, \dots, *), (*, \dots, *))$

-  $c_i$  := the central node of  $Q_i$

-  $L[c_i] := \{v\}$

- if  $c_i$  is a branching node

then  $bn[I(c_i)] := v$

else  $nbn[I_{nb}(c_i)] := v$

-  $E := \{(bn, nbn)\}$

- for each non-central node  $x$  in  $Q_i$  do

begin

-  $E' := \emptyset$

- for each  $e$  in  $E$  do

- for each  $u$  in  $L[x]$  do

begin

- create a copy  $e' = (bn', nbn')$  of  $e$

- if  $x$  is a branching node

then  $bn'[I(x)] := u$

else  $nbn'[I_{nb}(x)] := u$

- insert  $(bn', nbn')$  in  $E'$

end

-  $E := E'$

end

- for each embedding  $e = (bn, nbn)$  in  $E$  do

- emit( $[Q_i, (bn, nbn)]$ )

- for each  $(x, Q_j)$  in  $NBL$  do

- if  $x$  is a node in  $Q_i$  then

- for each  $u$  in  $L[x]$  do emit( $[Q_j, (x, u)]$ )

end

end.

*Example 7.* (Continued from Example 6).

The *reducer* with key  $(Q_1, Article1)$  receives the list of values

$[(n_1, Person4), (n_6, "Title1"), (n_1, Person1), (n_1, Person2), (n_3, Journal1)]$ . It constructs the lists:  $L_{n_1} = \{Person1, Person2, Person4\}$ ,  $L_{n_3} = \{Journal1\}$ ,  $L_{n_6} = \{"Title1"\}$ . It emits the following *key, value* pairs:

$Q_1, (\langle Person1, *, Journal1 \rangle, \langle Article1, *, Title1, * \rangle)$   
 $Q_1, (\langle Person2, *, Journal1 \rangle, \langle Article1, *, Title1, * \rangle)$   
 $Q_1, (\langle Person4, *, Journal1 \rangle, \langle Article1, *, Title1, * \rangle)$   
 $Q_3, (n_3, Journal1)$

The *reducer with key*  $(Q_2, Article1)$  receives the list of values  $[(n_1, Person4), (n_1, Person1), (n_1, Person2), (n_3, Journal1)]$ . It constructs the lists:  $L_{n_1} = \{Person1, Person2, Person4\}$ ,  $L_{n_3} = \{Journal1\}$ ,  $L_{n_7} = \{\}$ . Nothing is emitted.

The *reducer with key*  $(Q_2, Article2)$  receives the list of values  $[(n_1, Person2), (n_1, Person3), (n_3, Journal1), (n_7, "2008")]$ . It constructs the lists:  $L_{n_1} = \{Person2, Person3\}$ ,  $L_{n_3} = \{Journal1\}$ ,  $L_{n_7} = \{"2008"\}$ .

It emits the following *key, value* pairs:

$Q_2, (\langle Person2, Article2, Journal1 \rangle, \langle *, *, *, "2008" \rangle)$   
 $Q_2, (\langle Person3, Article2, Journal1 \rangle, \langle *, *, *, "2008" \rangle)$   
 $Q_1, (n_2, Article2)$   
 $Q_3, (n_3, Journal1)$

The *reducer with key*  $(Q_3, Person4)$  receives the list of values  $[(n_2, Article1), (n_2, Article3)]$ . It constructs the lists:  $L_{n_1} = \{\}$ ,  $L_{n_2} = \{Article1, Article3\}$ . Nothing is emitted.  $\square$

## 6.4 Phase 2 of the algorithm

Phase 2 of the algorithm is similar to the Phase 2 of the algorithm proposed in [10].

### 6.4.1 Mapper of Phase 2

Each mapper gets as input all the embeddings of a specific subquery  $Q_i$ ; moreover for each branching node that does not occur in  $Q_i$  it gets as input the values assigned to this node by the embeddings of the other queries. It fills in their missing branching node values using the corresponding values in the input, and emits the resulted embeddings to the reducers of Phase 2. The key is the tuple of the branching node values, which implies that two embeddings are emitted to the same reducer if and only if they are compatible.

```
mapper2( $Q_i$ , values)
//  $Q_i$ : the ID of a subquery
// values: a set  $E$  of the parts (bn, nbn) of the embeddings
//   of  $Q_i$  and a set  $V$  of pairs  $(n_k, v)$ ,
//   where  $v$  is a candidate value for  $bn[k]$ 
begin
- for each embedding  $e = (bn, nbn)$  in  $E$  do
- for each instance  $bn'$  of  $bn$  using the values in  $V$  do
- emit( $[bn', (Q_i, nbn)]$ )
end.
```

*Example 8.* (Continued from Example 7). The mapper that works for the subquery  $Q_1$ , gets a list of values that contain the embeddings of  $Q_1$  in  $G$ :

$Q_1, (\langle Person1, *, Journal1 \rangle, \langle Article1, *, Title1, * \rangle)$   
 $Q_1, (\langle Person2, *, Journal1 \rangle, \langle Article1, *, Title1, * \rangle)$   
 $Q_1, (\langle Person4, *, Journal1 \rangle, \langle Article1, *, Title1, * \rangle)$

and the values of missing branching nodes:

$(n_2, Article1), (n_2, Article2), (n_2, Article3)$

It emits the following *key, value* pairs to the reducers of Phase 2:

$\langle Person1, Article1, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person1, Article2, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person1, Article3, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person2, Article1, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$

$\langle Person2, Article2, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person2, Article3, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person4, Article1, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person4, Article2, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$   
 $\langle Person4, Article3, Journal1 \rangle, (Q_1, \langle Article1, *, Title1, * \rangle)$

The mapper that works for the subquery  $Q_2$ , receives a list of values containing the following embeddings

$Q_2, (\langle Person4, Article3, Journal2 \rangle, \langle *, *, *, "2008" \rangle)$   
 $Q_2, (\langle Person2, Article2, Journal1 \rangle, \langle *, *, *, "2008" \rangle)$   
 $Q_2, (\langle Person3, Article2, Journal1 \rangle, \langle *, *, *, "2008" \rangle)$

Notice that  $Q_2$  has no missing branching nodes. The mapper emits the following *key, value* pairs to the reducers:

$\langle Person4, Article3, Journal2 \rangle, (Q_2, \langle *, *, *, "2008" \rangle)$   
 $\langle Person2, Article2, Journal1 \rangle, (Q_2, \langle *, *, *, "2008" \rangle)$   
 $\langle Person3, Article2, Journal1 \rangle, (Q_2, \langle *, *, *, "2008" \rangle)$

The mapper that works for the subquery  $Q_3$ , get a list of values that contain the embeddings of  $Q_3$  in  $G$ :

$Q_3, (\langle Person4, Article1, * \rangle, \langle *, Person1, *, * \rangle)$   
 $Q_3, (\langle Person2, Article2, * \rangle, \langle *, Person3, *, * \rangle)$

and the values of missing branching nodes:

$(n_3, Journal1), (n_3, Journal2)$

It emits the following *key, value* pairs to the reducers:

$\langle Person4, Article1, Journal1 \rangle, (Q_3, \langle *, Person1, *, * \rangle)$   
 $\langle Person4, Article1, Journal2 \rangle, (Q_3, \langle *, Person1, *, * \rangle)$   
 $\langle Person2, Article2, Journal1 \rangle, (Q_3, \langle *, Person3, *, * \rangle)$   
 $\langle Person2, Article2, Journal2 \rangle, (Q_3, \langle *, Person3, *, * \rangle)$   $\square$

### 6.4.2 Reducer of Phase 2

Each reducer gets as input embeddings for each sub-query that are compatible (each one of them assigns the values in the key of the reducer to the branching nodes of the query). The embeddings (one for each subquery in  $(Q_1, \dots, Q_n)$ ) are joined to construct the final answers of  $Q$ :

```
reducer2(key, values)
// key: a tuple of branching node values
// values: pairs of the form
//   ( $Q_i$ , partial embedding for non-branching nodes)
begin
- for each join obtained by using one
  embedding for each subquery do
- Emit the result produced by this join
end.
```

*Example 9.* (Continued from Example 8). The reducer with key  $\langle Person2, Article2, Journal1 \rangle$  receives the list of values  $[(Q_1, \langle Article1, *, Title1, * \rangle), (Q_2, \langle *, *, *, "2008" \rangle), (Q_3, \langle *, Person3, *, * \rangle)]$  and constructs the unique embedding of  $Q$  in  $G$ :

$\langle Person2, Article2, Journal1 \rangle,$   
 $\langle Article1, Person3, Title1, "2008" \rangle$

The remaining 11 reducers do not return any answer (they don't receive values for at least one subquery).  $\square$

## 6.5 Implementation of the algorithm and experimental results

In this section, we present a set of preliminary experiments performed on a Hadoop cluster of 14 nodes of the following characteristics: Intel Pentium(R) Dual-Core CPU E5700 3.00GHz with 4GB RAM. We used five different datasets of sizes 113.6MB, 231.6MB, 491.5MB, 1.2GB, and 2.5GB obtained and adapted from the Lehigh University Benchmark (LUBM)<sup>2</sup>. These data sets are partitioned into

<sup>2</sup><http://swat.cse.lehigh.edu/projects/lubm/>

four, nine or fourteen data segments in correspondence with the number of the nodes of the cluster used in the experiment. Data segments are stored in different nodes of the cluster, in relational MySQL databases, whose schema consists of two tables one containing the triples of the segment and the other containing the border nodes.

The results of the first experiment are depicted in Table 1, where we can see that the algorithm is scalable in terms of the dataset size. In addition, the degree of the star (i.e. the number of the triples in the star) also affects the execution time: the larger the degree the smaller the execution time. The results of the second experiment (depicted in Table 2)

Dataset size	stars of degree 6	Stars of degree 3	Stars of degree 2	Num of Answers
113.6MB	190	219	254	93
231.6MB	212	258	280	189
491.5MB	310	369	459	402
1.2GB	650	689	727	999
2.5GB	1149	1200	1261	2007

**Table 1: Execution times (in seconds) for three different star-decompositions of a query in three different datasets, using a cluster of 14 nodes.**

show that our algorithm scales well by increasing the number of nodes in the cluster. In this experiment, the same queries have been computed (using the same star-decompositions) in 4, 9 and 14 computer nodes, for a fixed data set of size 231.6MB. Finally, the results of third experiment, depicted

# nodes	stars of degree 6	stars of degree 3	stars of degree 2
4	278	322	343
9	271	309	316
14	212	258	280

**Table 2: Execution times (in seconds) for three different star-decompositions of a query in a dataset of size 231.6MB, using clusters of 4, 9, and 14 nodes.**

in Table 3, show that the algorithm of this paper performs better in most cases (depending on the form of the given query) than the algorithm proposed in [10].

Alg.	query1	query2	query3	query4	query5
[10]	8,24	9,44	9,03	10,37	5,06
this	4,27	5,45	5,25	11,53	5,59

**Table 3: Execution times (in minutes) of two algorithms for five different queries, in a dataset of size 491,5MB, using a cluster of 14 nodes.**

## 7. CONCLUSION

In this paper, we present a two-phase MapReduce algorithm, for querying large amount of linked data, that extends our approach in [10]. The input query is decomposed into star subqueries and the answers to these subqueries are computed and joined to obtain the answers to the given query. Experimental evaluation shows that the algorithm is scalable in terms of the size of the data graph as well as the number of nodes in the cluster. In the near future we plan to compare the performance of the algorithm for different methods for decomposition of the input query into stars.

## 8. REFERENCES

- [1] Apache Hadoop Project. <http://hadoop.apache.org/>.
- [2] F. N. Afrati, D. Fotakis, and J. D. Ullman. Enumerating subgraph instances using map-reduce. In *ICDE*, pp. 62–73, 2013.
- [3] F. N. Afrati and J. D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.
- [4] C. Basca and A. Bernstein. Avalanche: Putting the spirit of the web back into semantic web querying. In *CEUR Workshop*, 2010.
- [5] M. Bröcheler, A. Pugliese, and V. S. Subrahmanian. A budget-based algorithm for efficient subgraph matching on huge networks. In *ICDE Workshops*, pp. 94–99, 2011.
- [6] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science and Engg.*, 11(4):29–41, 2009.
- [7] P. Cudré-Mauroux, I. Enchev, S. Fundatureanu, P. T. Groth, A. Haque, A. Harth, F. L. Keppmann, D. P. Miranker, J. Sequeda, and M. Wylot. Nosql databases for rdf: An empirical evaluation. In *International Semantic Web Conference (2)*, pp. 310–325, 2013.
- [8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
- [9] J.-H. Du, H. Wang, Y. Ni, and Y. Yu. HadoopRDF: A scalable semantic data analytical engine. In *ICIC (2)*, pp. 633–641, 2012.
- [10] M. Gergatsoulis, C. Nomikos, E. Kalogeros, and M. Damigos. An algorithm for querying linked data using map-reduce. In *Globe*, pp. 51–62, 2013.
- [11] J. Huang, D. J. Abadi, and K. Ren. Scalable SPARQL querying of large RDF graphs. *PVLDB*, 4(11):1123–1134, 2011.
- [12] M. F. Husain, L. Khan, M. Kantarcioglu, and B. M. Thuraisingham. Data intensive query processing for large RDF graphs using cloud computing tools. In *CLOUD*, pp. 1–10. IEEE, 2010.
- [13] P. Mika and G. Tummarello. Web semantics in the clouds. *IEEE Intelligent Systems*, 23(5):82–87, 2008.
- [14] J. Myung, J. Yeon, and S. goo Lee. SPARQL basic graph pattern processing with iterative mapreduce. In *MDAC*, ACM, 2010.
- [15] Z. Nie, F. Du, Y. Chen, X. Du, and L. Xu. Efficient SPARQL query processing in mapreduce through data partitioning and indexing. In *APWeb*, pp. 628–635, 2012.
- [16] N. Papailiou, I. Konstantinou, D. Tsoumakos, and N. Koziris. H2RDF: adaptive query processing on RDF data in the cloud. In *WWW (Companion Volume)*, pp. 397–400. ACM, 2012.
- [17] B. Quilitz and U. Leser. Querying distributed rdf data sources with sparql. In *ESWC*, pp. 524–538, 2008.
- [18] A. Schätzle, M. Przyjacieli-Zablocki, and G. Lausen. Pigsparql: mapping SPARQL to pig latin. In *SWIM*, 2011.
- [19] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *WWW*, pp. 607–614, 2011.

# Performance optimization for querying social network data

Florian Holzschuher  
iisys – Hof University  
Alfons-Goppel-Platz 1,  
95183 Hof, Germany  
+49 9281 409 6214  
florian.holzschuher@iisys.de

René Peinl  
iisys – Hof University  
Alfons-Goppel-Platz 1  
95183 Hof, Germany  
+49 9281 409 4820  
rene.peinl@iisys.de

## ABSTRACT

In this paper, we report about benchmark experiments and results from optimizing database connectivity for querying social networking data from Apache Shindig in a Neo4j database. We built on our experiments from [1] and tried to improve performance of the current RESTful http connection in comparison to JDBC in order to fully utilize performance benefits of the graph database compared to relational database management systems. We implemented a database driver based on WebSockets. We found that BSON is a better data transfer format than JSON and compression increases performance in some settings while decreasing it in others. Multiple WebSocket connections are needed to scale to a high number of client requests and fully utilize database performance. Multi-threading is another key factor for scalability. Implementing a kind of stored procedure, we were able to further increase throughput and decrease response times.

## Categories and Subject Descriptors

H.2.4 [Database Management]: Systems - Query processing

## General Terms

Performance, Experimentation.

## Keywords

Graph query processing, social networks, performance optimization, WebSocket, graph database

## 1. INTRODUCTION

Graph databases are a viable alternative to relational systems and perform especially well in domains like chemistry, biology and social networking [9]. In [1] we proved Neo4j to be a superior database backend for Apache Shindig compared to the existing JPA backend and MySQL. However, it seemed that RESTful http connections between client and server perform much worse than the TCP-based, permanent JDBC connection for JPA. RESTful http is a common choice for a NoSQL database, since it facilitates access from all programming languages that are able to use http and you don't have to provide drivers for every single language. CouchDB, Riak and AllegroGraph are examples of NoSQL databases using REST as their primary interface (see nosql-database.org). On the other hand, there had to be good reasons for computer engineers some ten years ago to put considerable efforts into connection pooling and similar optimization strategies for JDBC and other database connectivity technology [12, 13, 14].

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073).

Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

Therefore, we decided to proceed with our performance analysis and investigate different options for connecting the Neo4j backend to Shindig with a WebSocket-based driver (see Figure 1).

Our goal was to identify performance tuning factors for the graph database connection, while keeping the graph query language itself stable.

The remainder of the paper is structured as follows. We first discuss related work, especially other benchmarking approaches for graph databases of the last two years. Then we present the benchmark setup, discuss the relationship to previous results and compare performance of our WebSocket approach to embedded Neo4j and Cypher over RESTful http. We continue exploring the impact of different data transport formats and compression on performance and perform a detailed analysis of time measurements. As a last step, we present results from multi-threading, clustering and multiple connections before discussing limitations, future work and finishing with a conclusion.

## 2. Benchmark setup

Sample data and queries were the same as in our first published benchmarks in [1]. To briefly sum up, our sample data set covers a typical Web 2.0 intranet social networking portal and contains 2011 people, 26,982 messages, 25,365 activities, 2000 addresses, 200 groups and 100 organizations. The XML file generated is 45 MB in size and contains 1.5 million lines of text. Parsed into Neo4j, this set generates around 83,500 nodes and about 304,000 relationships, consuming just over 40 MB of disk space. On average, a person has 25 friends, at least 1 and a maximum of 667 resulting in about 25,000 friendship relations in total. 90% of people have less than 65 friends whereas the median is at 12 friends. We did not use the larger datasets with more people, activities and messages used in [1], since our tests generated enough data already and no significant differences were expected. We used the same 19 queries as in [1]. They are described briefly and categorized in the appendix. Due to space restrictions, we limited the diagrams to a subset.

In contrast to our first paper [1] we did not use VMs but physical hardware this time. The client with Apache Shindig was running on a server with AMD Opteron 870 CPUs (2 GHz) with 8 cores altogether and 32GB DDR RAM (400 MHz). Neo4j was running on one to five nodes with Intel Xeon X5355 CPUs (2,66 GHz) with 8 cores altogether and 32GB DDR2 RAM (667 MHz). All servers had a Gigabit network connection and a RAID 0 hard disk, but benchmarks ran completely in memory due to a warm-up that filled the caches. Monitoring data confirmed that there was less than one disk I/O operation per second in all benchmarks.

Our software consists of a client and a server part with a WebSocket connection in between (see Figure 1). Our benchmark client is based on Apache Shindig 2.5u1, the OpenSocial reference implementation, and creates the queries (step 1). This step also includes serialization of the query. This serialized query is

then transferred to the server using a pre-existing WebSocket connection (step 2). The connection is permanent and does not have to be established and closed for every single query, which is a major improvement compared to RESTful http. The server receives the query, deserializes it and executes it against the embedded Neo4j server (step 3). The driver (server part) and Neo4j database are running within the same process.

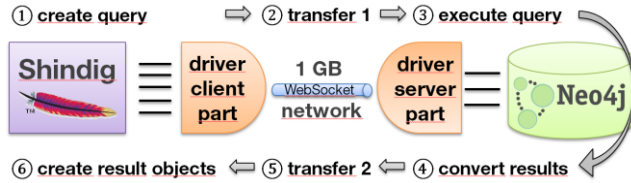


Figure 1: process of query execution

Once results from Neo4j are available, they are converted into a transferable structure (step 4). This step does not include serialization, but mainly consists of fetching additional data, since Neo4j always uses lazy loading of results. We are forcing the database to load all data, before transferring it to the client. Results are then serialized and transferred back over the network (step 5). These two steps were initially performed by the WebSocket library in one call that does both transmission and serialization using a previously defined converter class. We separated this later on in order to make better use of multi-threading for serialization. Finally, results are converted from the system independent transfer format to Shindig’s object structure (step 6).

We were using Neo4j 1.9.4 as a graph database, OpenJDK 7u25 as a runtime environment, Glassfish Tyrus WebSocket library 1.2.1, json.org 20090211 and MongoDB BSON serializer 2.11.2. All servers were running Ubuntu 12.04 LTS 64 bit.

We measured response times with `System.nanoTime()`. “On modern hardware and operating systems, it can deliver accuracy and precision in the microsecond range. Conclusion: for benchmarking, always use `System.nanoTime` ...” [9]. This is important, since many of our measured values are in the range of one millisecond or even below. One reason for moving from VMs to physical servers is the reliability of this measuring instrument, which only seems given for physical hardware. In order to collect network load, CPU load and memory usage, we used Monitorix<sup>1</sup> and modified it in order to increase time resolution from one minute to five seconds, since some of our tests only ran for two minutes.

### 3. Related work

In [1], we reported about our data generator for social networking data and performance comparison of several query languages.

Although we did not pay so much attention to correlation, our graph data generator follows a similar approach as [17]. We used dictionaries with real names, geographies, friendship networks and groups [1]. However, we did not intend to create big data at TB scale, but concentrate on intranet scenarios, i.e. medium to large organizations with a few thousand employees.

Although we did not crawl the data, but used the Stanford Large Network Dataset Collection as a basis instead, we followed a procedure similar to [7]. We also created a subset from a larger amount of available data that has no references regarding friendship or authorship pointing to entities outside the dataset. In addition to tweets (which we call messages), we are using activities like “Person *x* commented document *y* in System *z*”, or “Person *a*

rated activity *b* with three stars in System *c*”. We also coincidentally use the same number of queries (19) for benchmarking, although those of [7] are analytical queries, whereas our own are operational queries used in Shindig. They are for example used to display the user profile of a person, display an activity stream or suggest interesting colleagues for “friendship” formation. [7] roughly classify queries in three categories, i.e. “social network queries, timeline queries, and hotspot queries”. We had something similar and titled our query categories after the respective Shindig services *group*, *person*, *message*, *activity* and *graph service*. Graph service might be a misleading name, since all our queries are graph-oriented. These queries are for friend-of-a-friend (foaf) display, detection of shortest path between two people as well as friend and group suggestions. However, this doesn’t seem to describe queries well enough. [2] go further than that as part of the LDBC project and perform classification based on query characteristics. They introduce the categories *select*, *adjacency*, *reachability*, *summarization* and *pattern matching*. This seems to enhance traceability and we therefore tried to categorize our queries in the same manner (see appendix). However, assignment is not always clear since several queries have more than one of the properties described in one category. [3] also classify queries regarding basic operations involved and name especially a) point reads (based on primary key), b) CRUD operations based on primary key, c) association range queries for ID, type and timestamp range ordered from latest to oldest and d) association count queries, e.g. number of friends. Our own query mix includes (a), e.g. selective message read, (c), e.g. people’s friends activities and (d), although we usually fetch friend count together with top *x* friends. We do not benchmark write, update or delete operations. The most extensive classification is suggested in [5]. The authors present a multi-dimensional classification scheme describing the starting point (scope), reach (radius) and result of a query. We classified our queries regarding those criteria in order to make them more traceable. The result can be seen in the appendix.

We already reviewed some older graph benchmarks in [1]. [2] also benchmark Neo4j and conclude that it performs well, although a bit slower than Dex and usage of Cypher would be a viable option since it scales similarly well as the native API.

Another benchmark comparison between Neo4j and Dex is reported in [10], but they mainly use micro operations like “get vertex” or “get edge” instead of more complex queries. They found out that Neo4j scales very well for in-memory graphs, which is the case in our benchmark, but significantly loses performance when reading from disk and especially writing due to ACID transaction guarantees. They further mention that access of properties is considerably slower than access of vertices and edges for both Neo4j and Dex. We are accessing both vertices and properties in our benchmark.

[6] focus their benchmark on graph traversal operations and force the systems to perform disk I/O due to limited memory resources. They also use a graph data generator (LFR) and compare Dex, Neo4j, and four other systems. Neo4j performed well in breadth-first search with response times that are quite stable at less than 7,000 ms up to network sizes of 100,000 vertices, whereas Dex needed 15,000 ms for 10,000 vertices already. Computation of connected components on the other hand is scaling much worse, since response times increase dramatically for network sizes larger than 40,000 vertices.

A last study dealing with performance of graph databases in general and Neo4j in particular analyzes a special kind of data, namely social networking data varying over time [4]. They present a

<sup>1</sup> <http://www.monitorix.org/>



detailed data model and test ten queries. Neo4j performs well in eight of those and needs around 2,300 ms, due to a highly connected graph with up to 20,000 edges per vertex. These results are similar to ours, although distribution is even more extreme, since our slowest queries ran over 10 seconds and fastest under one ms.

#### 4. Comparison Cypher REST vs. Cypher WS

We first discuss differences between new results for Cypher over RESTful http and embedded benchmarks compared to the previous ones published in [1], before presenting the improvements of our WebSocket implementation.

##### 4.1 Relation to previous results

Although we tried to modify as few things as possible, the update from Neo4j 1.8 to 1.9 together with moving from VMs to physical servers influenced results. Keeping that in mind when comparing them, we still see Cypher performance improvements claimed by the vendor and anticipated in our previous paper [1] in the embedded benchmarks. In Figure 2, we show the results of the new benchmarks in relation to the respective previous ones (100%). Cypher needs only 57% of the time for running our queries compared to the results previously published. Native implementation also gains in most cases, although there are a few exceptions, where performance loss of 13% and 29% respectively can be noticed. The median is still 69%, which means that native implementation is roughly 30% faster than before.

For Cypher, results get even better, since embedded Cypher outperforms native implementation in our tests with multiple threads already at 16 threads with 1343 requests per second and reaches a maximum of 1370 req/sec with 64 threads, whereas native implementation reaches its maximum at 1323 req/sec with 128 threads. This is especially interesting, since native implementation is more than 30% faster for single threads.

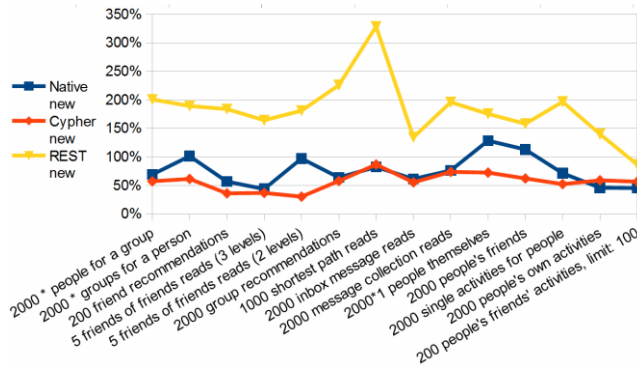


Figure 2: comparison of new results with previous ones

The downside is that although Cypher performs better than before, the connection over RESTful http got worse by nearly factor two. This further encouraged us to move on with our own connection library that should provide significant performance benefits. Part of the performance loss could be caused by the slower physical network connection between servers compared to the purely virtual connection on the VMs in our previous experiments, although our Gbps network connection's capacity was never fully utilized.

##### 4.2 WebSocket performance

In this section, we discuss performance of Cypher queries over WebSocket with JSON (Cypher JSON) compared to Cypher over RESTful http with JSON (Cypher REST). We also introduced a kind of stored procedure, where the client only calls the procedure by name and passes parameters along (Native JSON). On the

server, a native procedure is then executed. This is a higher implementation effort, but might prove worthwhile for single queries like foaf where Cypher still does not perform very well. Stored procedures using predefined Cypher queries could hardly be used due to the dynamic nature of Shindig requests. Therefore, cypher-rs<sup>2</sup>, a Neo4j server extension for stored Cypher queries was no option. Figure 3 shows the results in relation to http performance (100%). Absolute query times for Cypher REST lie in between two and 20 seconds or in between four and 100 ms broken down to single retrievals. However, there are three exceptions, namely friend recommendations and both friend of a friend tests (foaf). Due to Cypher language constraints or inefficient implementation these queries take up to 426 seconds or between 367 and 21,330 ms on the single query level. We therefore consider them to be spikes and discuss them separately. These spikes were already present in our last test [1], although there were some performance improvements for Cypher (see section 4.1).

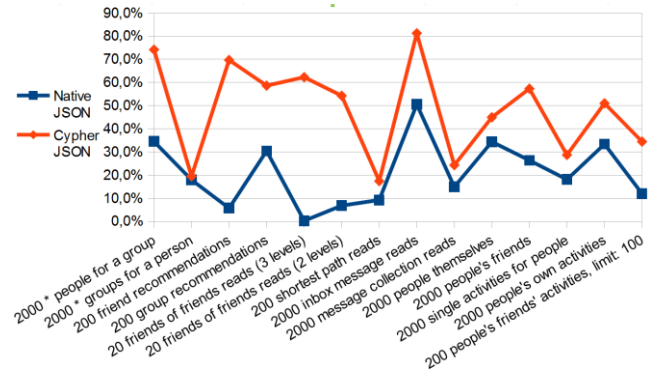


Figure 3: comparison of WebSocket and REST performance

Cypher JSON is faster than Cypher REST in all cases. Performance gains range from 83% for shortest path until 9% for inbox message reads and average at 40-50%. This is fairly good, since we were only enhancing the network connection between client and server and not the queries. However, we were inspired by the idea of stored procedures in relational database systems as mentioned above and wanted to further explore such a possibility in Neo4j. On the server, you can register your own Java implementations of such queries using Google Guice's injection mechanisms without recompiling the project. The afore mentioned spikes are good candidates for those stored procedures and the native JSON line in Figure 3 shows that performance in these cases is about ten times higher than RESTful http and even 200 times higher for foaf with three levels. A mix of normal Cypher for most of the queries, with these three "stored procedures" increased performance gains from about factor two to factor nine, which seems impressive and worth the effort.

#### 5. Performance tuning

After these initial results, where only network connection was changed from pure http to WebSocket, we decided to investigate the impact of different choices regarding data transfer format (5.1), compression (5.2) as well as conversion from Neo4js' object model into Shindigs object model (5.3).

##### 5.1 JSON vs. BSON

For the primary transfer format, we considered binary JSON (BSON) as an alternative to JSON, since it is type safe and we didn't need the better compatibility of JSON, since we control

<sup>2</sup> <https://github.com/jexp/cypher-rs>



both ends of the connection. Contrary to intuition, the binary BSON format produces slightly larger objects than the text-based JSON due to additional metadata regarding data types. Figure 4 shows the relative response times of our queries with BSON serialization in relation to its JSON counterparts (100%).

Native implementation gains more from using BSON instead of JSON although Cypher usually produces larger result sets and the full query in Cypher language has to be transferred instead of the name of the “stored procedure” together with parameter values in the native implementation. Therefore, one would expect that the serialization format has a larger impact there. However, native implementation is much faster than Cypher and therefore the relative impact of serialization benefits is much higher. The median for performance improvements of BSON is 44% for native implementation and 36% for Cypher. Single queries are up to three times faster for native implementation. We therefore consider BSON to be the better choice and concentrate our further results on this option.

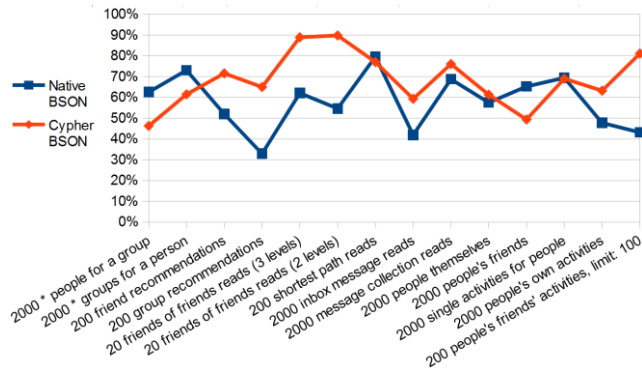


Figure 4: response time of BSON in relation to JSON

## 5.2 Compression

Since we are optimizing network transmission and conversion infrastructure, compression could be an interesting option. However, our WebSocket implementation does already transmit data much more efficient than the original RESTful http connection. In one of our scenarios, we measured 70 MByte/sec network load on the client for RESTful http, whereas our WebSocket implementation needed 17.4 MByte/sec only and achieved 3.8 times the number of requests/sec (see section 7.1), so that it is factor 15 more efficient. We tested the built-in Java zlib deflate algorithm with fast and best compression settings. With fast compression we were able to reduce network bandwidth usage further to 5.4 MByte/sec – again factor 3 more efficient. Best compression only marginally further reduced network load to 4 MByte/sec, but at the expense of slower operation and higher CPU load.

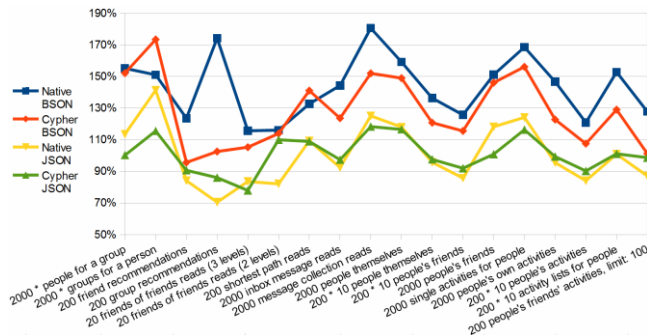


Figure 5: response time of fast in relation to no compression

Figure 5 depicts relative response times for single threaded queries and fast compression compared to no compression (100%). For BSON, fast compression achieved significantly slower answer times (30% slower for Cypher, 50% for native). In the JSON case some lower answer times and some higher times compensate more or less for each other for fast compression. Best compression performed 10% worse on average.

Astonishingly, compression is still useful in some scenarios with high throughput, although the Gbps network card is far from saturated, so that network traffic shouldn't be the bottleneck. We assume that the single threaded WebSocket implementation is the limiting factor (see section 7.3). Summed up, for 128 threads on the client and 8 threads on the server, fast compression achieved 64% more throughput and even best compression was 45% faster than no compression (see section 7 below).

## 5.3 Object model conversion and serialization

One annoying thing about querying the database is type conversion. With JDBC, there is a similar problem with dates and times (`java.util.Calendar` vs. `java.sql.Date` or `java.sql.Timestamp`). In our case the problems were arrays and lists. Neo4j internally uses arrays for multi-value properties. Both Shindig and the BSON serializer however, use java lists exclusively, although data is transferred as a BSON array. The same applies to JSON. Therefore, we had to convert at multiple points from lists to arrays and back, which is an annoying overhead, although handled by a single call of `List.toArray()` and `Arrays.asList()` respectively.

Furthermore, we identified serialization and deserialization as a potential performance bottleneck. In our initial tests, we let Tyrus (the WebSocket library from Glassfish) handle serialization of Java objects, since it conveniently performs it as part of the transfer and only requires a serializer class for the desired message format to be registered. However, this process is largely single threaded and therefore posed a limitation for scaling as soon as we introduced multiple client threads. Therefore, we chose to handle serialization ourselves before transmitting raw data using Tyrus in order to make it multi-threaded. Each client thread is creating its own converter objects due to thread safety. We considered using a pool of conversion objects instead of constantly instantiating new ones, but haven't implemented it yet.

Although we are not able to present any reliable numbers on that particular implementation detail, we are sure that it is the foundation for scaling to a large number of client requests.

## 6. Detailed performance analysis

After completion of the experiments described above, it seemed that uncompressed BSON was the best option and we could not further enhance performance of the connection. However, we did not have the impression to fully understand which components represented the bottleneck limiting throughput, since neither CPU nor network were used to full capacity. Therefore, we tried to investigate further and come up with more detailed time measurements.

Single processing steps shown in Figure 1 could not be logged consistently in all detail. However, we managed to capture times for steps 1+2+6 (client processing time), steps 3+4 (server processing time) and step 5 (network time). Figure 6 shows results for some of our queries and compares JSON with BSON serialization as well as fast compression and no compression. Relative times are depicted in bar charts, whereas absolute times are shown in milliseconds. Numbers above the bars are total times for the individual type of query.

Time reductions for BSON compared to JSON are achieved in both serialization and deserialization, which is included in the client and the network times depicted in Figure 6. Server times are only marginally reduced, since only deserializing the query is part of this time measure, which requires less effort than deserializing the response, which is up to 500 kB in size for friend recommendation and up to 200 kB for FOAF responses.

Compression is slower in all three parts for BSON and is especially striking in the network times that nearly double due to compression of the server response that is included in this time. Faster transfer of the reduced message size is negligible in relation to compression time when using a Gbps network connection.

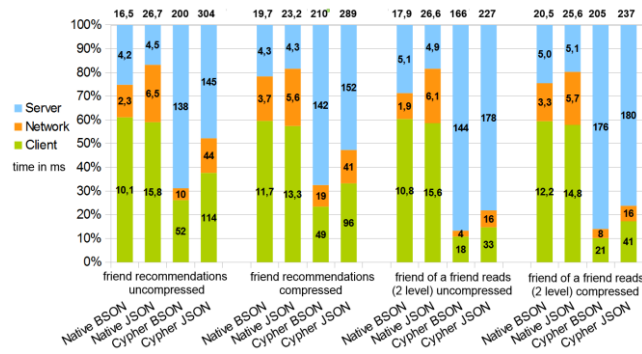


Figure 6: processing times split into client, server and net

## 7. Scaling tests

Having seen results for single threads, we additionally tested how threading on client and server impacted the results. We switched benchmarking measurements from time per request to requests per second. In order to better capture real relations, we limited queries to those that were not identified as spikes before (see section 4.2).

### 7.1 Threading

Initially, we used a single thread only in order to understand performance impacts of different options. Now, we wanted to explore the scalability with multiple threads. On the client side, we used 4, 16, 64 and 128 threads. On the server side we used powers of two up to 16. Figure 7 shows the results without compression. Numbers along the x-axis represent server threads.

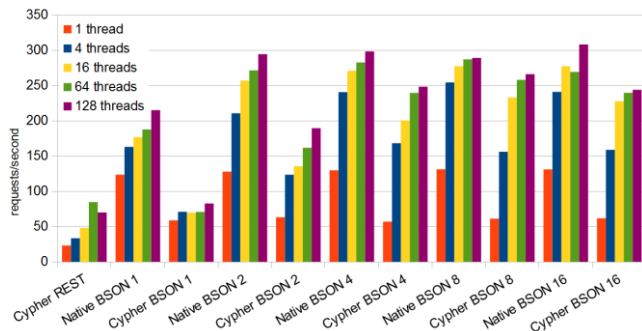


Figure 7: results with multi-threading in requests per sec.

For our native implementation the server didn't scale very well: only 37% for the step from one to two server threads and only 3% more for 16 threads. Cypher scaled better and gained an astonishing 229% for two threads but only additional 31% for the step to 4 threads. Afterwards, it gains only 7% for eight threads and even loses 8% for 16 threads. Since the server has 8 cores and neither CPU nor network are saturated, we suspected the comparably slow memory of the client and the single threaded WebSocket

implementation to be the bottleneck. The good thing about the different scalability of native and Cypher implementations is that Cypher BSON reaches 86% of the maximum throughput and outperforms the old Cypher REST by more than factor three. This performance gain is in a similar order of magnitude as reported in [11].

Although we assumed that compression could not provide performance improvements in our setup, we included it in our threading benchmarks. This led to surprising results, since fast compression already proved to be superior to no compression with a single server thread and reached about 66% higher performance for eight server and 128 client threads (see Figure 8). Scaling results per se are quite similar to uncompressed results, but at a slightly higher level. Native implementation only scales well up to two server threads and only marginally gains for more threads. Cypher performance improves by a surprising 112% for two threads and an additional 57% and 50% for four and eight threads. 16 threads did not increase throughput further, which is not surprising due to the fact that the server only has eight cores.

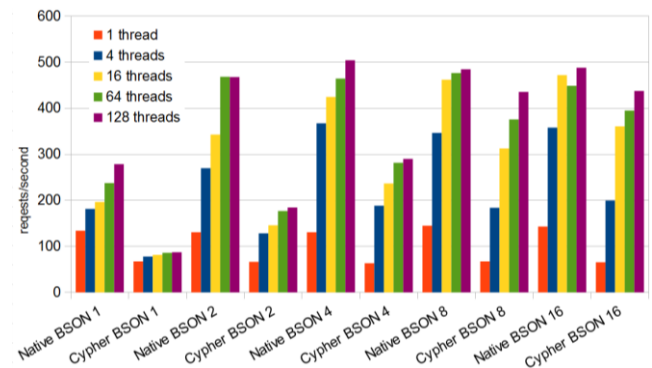


Figure 8: results with multi-threading and fast compression

### 7.2 Cluster

The next step was to move from a single server to use multiple Neo4j servers in a cluster. We implemented a cluster-enabled client driver that distributes requests evenly with a round robin algorithm. It works similarly to C-JDBC [12]. Based on our scaling tests (see section above) we configured the server to run with eight threads, which means one per core. This setup scales relatively well from one to three nodes, as can be seen in Figure 9.

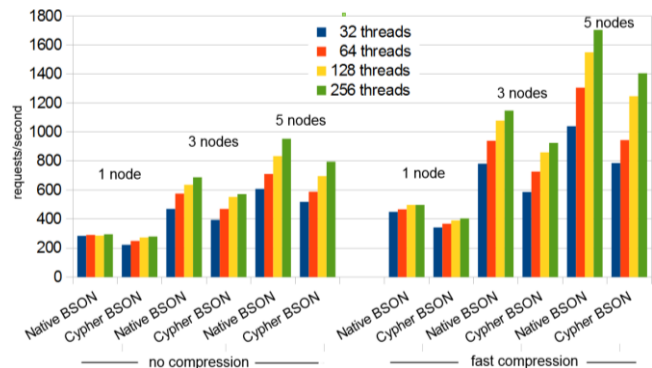


Figure 9: requests/sec with different cluster setups

Native implementation gains 133%, Cypher does not scale quite as well and reaches 105%. That still seems acceptable compared to the increase in hardware resources of 200%. Although exact numbers are not visible very well in the 3D diagrams presented in [11], it seems that our performance gains lie between MySQL and PostgreSQL with C-JDBC [11]. Performance gains for further

increasing hardware resources to five nodes are considerably lower with 39%, so that native implementation reaches 223% of single node throughput and Cypher 185%.

Compression turns out to be an important option in this scenario as well. Even with one node, throughput is between 43% (Cypher) and 74% (native) higher with fast compression than without compression. Maximum throughput with five nodes and 256 client threads is even 77% (Cypher) and 79% (native) higher than that without compression. This is mainly due to better scaling from three to five nodes. Whereas performance gains for moving from one to three nodes is similar for fast compression to no compression (131% for native and 129% for Cypher), the step from three to five nodes leads to another 48% (native) and 52% (Cypher) performance improvement for fast compression, which is significantly higher than the 39% for no compression.

### 7.3 Multiple WebSocket connections

We finally tried to use multiple WebSocket connections (conns) between client and server to get either CPU or network fully utilized. We directly aimed for eight connections and skipped tests for two and four, but varied the number of server threads per connection. Figure 10 shows results from one server node, 128 client threads, the depicted number of connections and server threads per connection. Results are shown for Cypher BSON and Native BSON without compression (left bar group) and fast compression (right bar group).

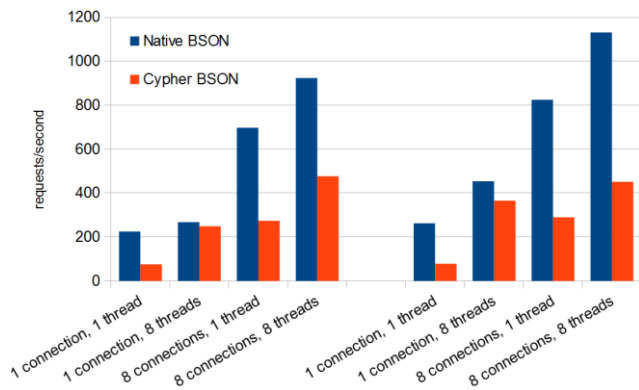


Figure 10: requests/sec with multiple WebSocket connections

It turned out, that this was indeed the limiting factor and none of our previous measurements had revealed that. Results for native implementation without compression, eight connections and eight server threads per connection outperforms both three node (45% higher throughput) and even five node cluster results (11% higher throughput) with one connection and otherwise identical settings. For fast compression at least the three node cluster is beaten by 5%. Unfortunately, Cypher does not benefit equally from multiple connections. Where native implementation gains impressive 245% when moving from one to 8 connections with 8 server threads (uncompressed), Cypher gains only 91% and therefore reaches only 52% of native throughput. This is due to CPU usage, as shown in Figure 11.

Small usage spikes at the beginning represent warm-up procedure. Then native tests starting briefly before 16:30 are utilizing all eight CPUs at roughly 60% with spikes up to 75%. Then again a warm-up is run and Cypher tests start around 16:32:30. CPU load is near 100% there. Looking at single CPU cores, we see that all except one core are saturated at 100% and the last one at 90%. This state is nearly reached for native implementation with fast compression, where global CPU load averages at 95%. Network

load is not the limit. It reaches 50 MByte/sec without compression and 12 MByte/sec for fast compression. We did not manage to run all the multi-connection tests in the cluster, but gave the most promising constellation a shot and achieved **2544 req/sec** for 3 nodes Native BSON and fast compression (2489 req/sec for 5 nodes) and 1824 req/sec for 5 nodes with Cypher BSON (1266 req/sec for 3 nodes). Results without compression were lower.

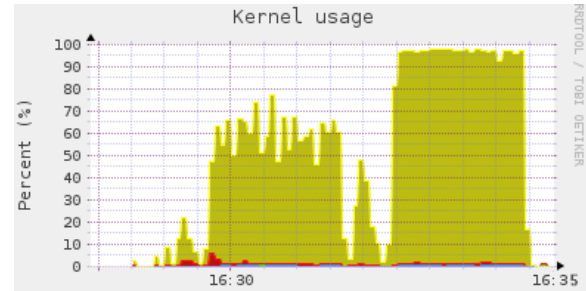


Figure 11: global kernel usage on the server (Monitorix)

That means that Cypher scales nearly linearly with the number of nodes (300% nodes => 281% performance, 500% nodes => 404% performance) with multiple WebSocket connections. This is better than the results for a cluster with a single WebSocket connection per server (see Table 1). Native implementation, on the other hand, reaches a limit at the three node cluster setup (225% performance) and does not scale further (220% for 5 nodes). The client was not able to issue more requests.

Table 1: throughput for cluster with single and multiple conns

	Native	Cypher
3 nodes, single conn	1.148 r/s (231%)	926 r/s (229%)
3 nodes, 8 conns	2.544 r/s (225%)	1.266 r/s (281%)
5 nodes, single conn	1.705 r/s (342%)	1.405 r/s (347%)
5 nodes, 8 conns	2.489 r/s (220%)	1.824 r/s (404%)

## 8. Limitations

Our benchmark queries are directly derived from Apache Shindig. However, some of them are only present in our extended version of Shindig that supports friend and group recommendations as well as shortest path analysis like many other social networks provide them (e.g. Xing). The single queries are not weighted based on frequency of use in normal user scenarios.

Furthermore, we did not make any efforts to optimize RESTful http, so that it uses one thread per request, which leads to some overheads. We did not consider using an asynchronous event-driven implementation instead [16] that might positively impact performance when carefully tuned [15]. We didn't test alternative JSON serialization libraries, although we were pointed to an interesting resource lately, showing very fast implementations<sup>3</sup>.

We also did not test all alternatives with multiple WebSocket connections, since some early tests with this feature proved unsuccessful so that we did not consider it for inclusion up to a few days before submission. When we remembered to retest it with our other improvements and physical machines it turned out to make a big difference. Therefore, we could only conduct the few tests discussed here instead of the full suite. Finally, it would have been desirable to have different hardware options in order to gain further insight into how different CPU speeds and cores affect

<sup>3</sup> <https://github.com/eishay/jvm-serializers/wiki>

overall performance or whether speed of system memory really is a limiting factor in some tests.

## 9. Future work

In parallel to our work described here, Neo Technology is finishing work on Neo4j 2.0. Keen on any enhancements the new version is providing, we did a preliminary test with Neo4j 2.0 M06 dating from 15<sup>th</sup> of October 2013. We found that due to introduction of multiversion concurrency control (MVCC), neither the native implementation nor Cypher queries ran without changes. We had to make significant changes and to introduce at least one transaction for every query. This leads to decreased performance in most of our tests. Native implementation loses the most with 19% in embedded and 30% in WebSocket BSON benchmark. For Cypher there have been further language optimizations so that some of the Cypher tests gained performance, most notably foaf (20-50%) as well as friend and group recommendation (20%-30%). Still, overall performance decreased here as well losing 12% in embedded and WebSocket and 7% in our REST benchmark. These results have to be interpreted with caution, since we did not use all new query features and did not optimize our queries and algorithms for the new version. All we did were changes to get queries running. There is e.g. a whole new transactional http API that finally gets rid of the superfluous URLs that were delivered with every result and led to the tremendous overhead reported in [1]. We have not investigated this new endpoint yet.

Being confident that we optimized the network connection quite well, we plan to further explore end-to-end performance of Shindig together with the database and use jMeter to query the Shindig REST API instead of using our own benchmarking tool. It could be the case that Shindig is not able to benefit from our optimizations due to own internal inefficiencies. Another influence could come from switching from running a standalone client and server to running them in Apache Tomcat and Glassfish.

We also plan to use much higher volumes of data, so that Neo4j has to access disks, instead of caching all data in main memory.

Finally, we aim at including benchmarks for writing data to Neo4j, since all our current queries are read-only. Neo4j provides single master replication within all nodes of an enterprise cluster, which we were using in our cluster tests. It will be interesting to see how well write operations scale and how fast replication between nodes really is.

## 10. Conclusion

We've presented results from optimizing the database connection to Neo4j for querying social networking data from Apache Shindig. We've thoroughly analyzed several options for the transfer including JSON vs. BSON as a data transfer format, different compression options, multiple WebSocket connections as well as multi-threading on client- and server-side for achieving the highest possible throughput. We then went from a single server database to a cluster of three and five nodes in order to analyze scalability. Results show that BSON is more efficient than JSON, especially regarding (de-)serialization. Compression reduces network load significantly and performs better for a high number of client requests. Multiple WebSocket connections increase maximum throughput significantly (up to 245%).

The database cluster is able to increase throughput and achieves a maximum of 181% performance increase for 200% additional server resources with Cypher. Scaling to five servers did not result in better throughput for native implementation. It might be the case that the client was the limiting factor there. Cypher however, was able to gain an additional 44% of performance compared to

three nodes which lead to a 304% performance increase altogether in comparison to a single node. Table 2 summarizes results of our test in relation to Cypher over RESTful http. Values in parentheses represent the performance relative to REST.

Summed up, we can state that it is worthwhile to pay attention to network connectivity between application server and database server. We are convinced that many NoSQL databases are experiencing similar problems causing them to not fully expose their internal performance over a standard RESTful http interface. It would be interesting to measure performance of our approach compared to RexPro and Rexster from the tinkerpops project, that aim at providing a kind of JDBC-like standardization to graph databases. With systematic analysis and consequent enhancement of our database driver, we were able to increase speed by factor 5.6 for Cypher and up to 13.3 when using our "stored procedures" that ran native queries on the database server. When considering extreme graph queries like three level foaf, performance differences are even higher.

Table 2. Summary of results

	REST	Native BSON	Cypher BSON
single thread response time	100%	17%	38%
128 threads throughput (r/s)	70	215 (3.1x)	83 (1.2 x)
Max 1 node throughput (r/s)	85	1131 (13.3x)	476 (5.6x)
Global max throughput (r/s)	85	2544 (29.9x)	1824 (21.5x)

Although the benchmark is specific to Apache Shindig, the Neo4j driver is generic and can be used in any project. As an intended side effect of our efforts, open source projects with a more liberal license like APL v2 can now use GPL v3 licensed Neo4j, without fear of the viral GPL, since a pure network connection separates client and server part of our driver, so that GPL does not affect client code. This is another major improvement compared to our efforts in [1]. Therefore, we hope that our Neo4j backend will soon become the default for Apache Shindig.

## 11. REFERENCES

- [1] Holzschuher, F., and Peinl, R. 2013. *Performance of graph query languages: comparison of cypher, gremlin and native access in Neo4j*. Joint EDBT/ICDT Workshop GraphQ 2013. Genoa, Italy. 22.03.2013. 195-204
- [2] Angles, R., Prat-Pérez, A., Dominguez-Sal, D., and Larriba-Pey, J. L. (2013) *Benchmarking database systems for social network applications*. 1<sup>st</sup> International Workshop on Graph Data Management Experiences and Systems. ACM. 15
- [3] Armstrong, T. G., Ponnkanti, V., Borthakur, D., and Callaghan, M. (2013) *LinkBench: a Database Benchmark Based on the Facebook Social Graph*. ACM SIGMOD, June 2013. 1185-1196
- [4] Cattuto, C., Quaggiotto, M., Panisson, A., and Averbuch, A. (2013) *Time-varying social networks in a graph database: a Neo4j use case*. In 1<sup>st</sup> International Workshop on Graph Data Management Experiences and Systems. ACM. 11
- [5] Grossniklaus, M., Leone, S., and Zäschke, T. (2013) *Towards a benchmark for graph data management and processing*. Technical Report KN-2013-DBIS-01, University of Konstanz, Department of Computer and Information Science

- [6] Ciglan, M., Averbuch, A., and Hluchy, L. (2012) *Benchmarking traversal operations over graph databases*. In Data Engineering Workshops (ICDEW 2012). 186-189. IEEE.
- [7] Ma, H., Wei, J., Qian, W., Yu, C., Xia, F., & Zhou, A. (2013) *On benchmarking online social media analytical queries*. In 1<sup>st</sup> International Workshop on Graph Data Management Experiences and Systems. ACM. 10
- [8] Boyer, B. (2008) *Robust Java benchmarking, Part 1- Understand the pitfalls of benchmarking Java code*. <http://www.ibm.com/developerworks/java/library/j-benchmark1/index.html>
- [9] Miller, J. J. (2013). *Graph Database Applications and Concepts with Neo4j*. Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA March 23rd-24th, 2013.
- [10] Macko, P., Margo, D., and Seltzer, M. (2013). *Performance introspection of graph databases*. In Proceedings of the 6th International Systems and Storage Conference. ACM. 18
- [11] Unde, P., Vin, H., Natu, M., Kulkarni, V., Thomas, D., Vasudevan, S., and Pathak, R. (2012) *Architecting the Database Access for a IT Infrastructure and Data Center Monitoring tool*. In IEEE 28th International Conference on Data Engineering Workshops (ICDEW), 2012. 351-354
- [12] Cecchet, E. (2004) *C-JDBC: a Middleware Framework for Database Clustering*. IEEE Data Engineering. Bulletin. 27(2), 19-26.
- [13] Karlsson, M., Moore, K. E., Hagersten, E., & Wood, D. A. (2003). *Memory system behavior of Java-based middleware*. In 9<sup>th</sup> int. Symposium on High-Performance Computer Architecture, HPCA-9 2003. (pp. 217-228). IEEE.
- [14] Li, Y., & Lü, K. (2000). *Performance issues of a Web database*. In Database and Expert Systems Applications (pp. 825-834). Springer Berlin Heidelberg.
- [15] Malkowski, S., Jayasinghe, D., Hedwig, M., Park, J., Kanemasa, Y., & Pu, C. (2010). *Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload*. In Proceedings of the 2010 ACM Symposium on Applied Computing. 1680-1687.
- [16] Harji, A. S., Buhr, P. A., & Brecht, T. (2012). Comparing high-performance multi-core web-server architectures. 5th Annual International Systems and Storage Conference (p. 2).
- [17] Pham, M.-D., Boncz, P. & Erling, O. (2012) S3G2: A Scalable Structure-Correlated Social Graph Generator. 4th TPC Technology Conference, Istanbul, Turkey, 27.08.2012

## Appendix: Query classification

Query	Description	Scope	Radius	Result	Type
2000 * people for a group	Group x => members	node	neighbors	nodes	select
2000 * groups for a person	Person x => group membership	node	neighbors	nodes	select
200 friend recommendations	Person x => friend => friend[not x's friend] sort by friends in common	node	neighbors	nodes	pattern matching
200 group recommendations	Person x => friend => group[not x's group] sort by num friends with this group	node	neighbors	nodes	pattern matching
20 friends of friends reads (3 levels)	Person x => friend => friend => friend [not x's friend]	node	neighbors	nodes	pattern matching
20 friends of friends reads (2 levels)	Person x => friend => friend [not x's friend]	node	neighbors	nodes	pattern matching
200 shortest path reads	Person x, y => shortest path between	path	path	subgraph	reachability
2000 inbox message reads	Person x => message collection[inbox] => messages	node	neighbors	nodes	select
2000 message collection reads	Person x => message collections incl. num messages per collection	node	neighbors	nodes	summarization
2000 people themselves (profile page)	Person x => all first level properties => some second level properties	node	neighbors	subgraph	adjacency
200 * 10 people themselves	Person a, b, ..., j => all first level properties => some second level properties	subgraph	neighbors	subgraph	adjacency
200 * 10 people's friends	Person a, b, ..., j => friends	subgraph	neighbors	nodes	adjacency
2000 people's friends	Person x => friends	node	neighbors	nodes	adjacency
2000 single activities for people	Person x => activities[a]	node	neighbors	nodes	select
2000 people's own activities	Person x => activities	node	neighbors	nodes	adjacency
200 * 10 people's activities	Person a, b, ..., j => activities	subgraph	neighbors	nodes	adjacency
200 * 10 activity lists for people	Person x => activities[a, b, ..., j]	node	neighbors	nodes	select
200 people's friends' activities, limit: 100	Person x => friends => activities => sort by created descending [1..100]	path	neighbors	nodes	adjacency



# Frequent Pattern Mining from Dense Graph Streams

Juan J. Cameron  
University of Manitoba, Winnipeg, MB, Canada  
umcame33@cs.umanitoba.ca

Fan Jiang  
University of Manitoba, Winnipeg, MB, Canada  
umjian29@cs.umanitoba.ca

Alfredo Cuzzocrea  
ICAR-CNR & Uni. Calabria, Rende (CS), Italy  
cuzzocrea@si.deis.unical.it

Carson K. Leung  
University of Manitoba, Winnipeg, MB, Canada  
kleung@cs.umanitoba.ca

## ABSTRACT

As technology advances, streams of data can be produced in many applications such as social networks, sensor networks, bioinformatics, and chemical informatics. These kinds of streaming data share a property in common—namely, they can be modeled in terms of graph-structured data. Here, the data streams generated by graph data sources in these applications are *graph streams*. To extract implicit, previously unknown, and potentially useful *frequent patterns* from these streams, efficient data mining algorithms are in demand. Many existing algorithms capture important streaming data and assume that the captured data can fit into main memory. However, problems arise when such an assumption does not hold (e.g., when the available memory is limited). In this paper, we propose a data structure called *DSMatrix* for capturing important data from the streams—especially, dense graph streams—onto the disk when the memory space is limited. In addition, we also propose two stream mining algorithms that use *DSMatrix* to mine frequent patterns. The tree-based *horizontal mining algorithm* applies an effective *frequency counting approach* to avoid recursive construction of sub-trees as in many tree-based mining. The *vertical mining algorithm* makes good use of the information captured in the *DSMatrix* for mining.

## Categories and Subject Descriptors

E.1 [Data]: Data Structures—*graphs and networks*; H.2.8 [Database Management]: Database Applications—*data mining*

## General Terms

Algorithms; Design; Experimentation; Management; Performance; Theory

## Keywords

Data mining, frequent pattern discovery, graph patterns, graph-structured data, social networks, extending database technology, database theory

## 1. INTRODUCTION & RELATED WORK

Since the introduction of the research problem of frequent pattern mining from traditional static databases [3], numerous studies [9, 18, 22] have been proposed. Examples include the Apriori algorithm [3]. To improve efficiency, Han et al. [16] proposed the FP-growth algorithm, which uses an extended prefix-tree structure called *Frequent Pattern tree (FP-tree)* to capture the content of the transaction database. Unlike Apriori that scans the database  $k$  times (where  $k$  is the maximum cardinality of the mined frequent patterns), FP-growth scans the database twice. Although there are some works [5, 15] that use disk-based structure for mining, they mostly mine frequent patterns from *static* databases. As a preview, we mine frequent patterns from *dynamic* data streams. When dealing with these streaming data, we no longer have the luxury of scanning the data multiple times, for instance in support of complex knowledge discovery processes from data streams, like *OLAP analysis over data streams* (e.g., [10]).

Over the past decade, the automation of measurements and data collection has produced high volumes of valuable data at high velocity in many application areas. The increasing development and use of a large number of sensors has added to this situation. These advances in technology have led to streams of data such as sensor networks, social networks, road networks [19, 28]. These kinds of data share in common the property of being modeled in terms of *graph-structured data* [23] so that the streams they generate are, properly, *graph streams* (i.e., streams of graphs). In order to be able to make sense of streaming data, stream mining algorithms are needed [17, 24, 26]. When comparing with the mining from traditional *static* databases, mining from *dynamic* data streams is more challenging due to the following properties of data streams:

*Property 1: Data streams are continuous and unbounded.* To find frequent patterns from streams, we no longer have the luxury of performing multiple data scans. Once the streams flow through, we lose them. Hence, we need some data structures to capture the important contents of the streams (e.g., recent data—because users are usually more interested in recent data than older ones (e.g., [12, 11])).

*Property 2: Data in the streams are not necessarily uniformly distributed; their distributions are usually changing with time.* A currently infrequent pattern may become frequent in the future, and vice versa. So, we have to be careful not to prune infrequent patterns too early; otherwise, we may not be able to get complete information such as frequencies of certain patterns (as it is impossible to retract

those pruned patterns).

Several approximate and exact algorithms have been proposed to mine frequent patterns from data streams. *Approximate* algorithms (e.g., FP-streaming [14], TUF-streaming [20]) focus mostly on efficiency. However, due to approximate procedures, these algorithms may find some infrequent patterns or miss frequency information of some frequent patterns (i.e., some false positives or negatives). An *exact* algorithm mines only truly frequent patterns (i.e., no false positives and no false negatives) by (i) constructing a *Data Stream Tree (DSTree)* [21] to capture contents of the streaming data and then (ii) recursively building FP-trees for projected databases based on the information extracted from the DSTree.

While the above two properties play an important role in the mining of data streams in general, they play a more challenging role in the mining of a special class of data streams—namely, *graph streams*. Nowadays, various graph data sources can easily generate high volumes of streams of graphs (e.g., direct acyclic graphs representing human interactions in meetings [13], social networks representing connections or friendships among social individuals [7, 25]). However, when comparing with data streams in general, graph streams in particular are usually more difficult to handle [4]. Problems and state-of-the-art solutions are highlighted in recent studies. For instance, Aggarwal et al. [2] studied the research problem of mining *dense patterns* in graph streams, and they proposed probabilistic algorithms for determining such structural patterns effectively and efficiently. Bifet et al. [4] mined *frequent closed graphs* on evolving data streams. Their three innovative algorithms work on coresets of closed subgraphs, compressed representations of graph sets, and maintain such sets in a batch-incremental manner. Moreover, Aggarwal [1] explored a relevant problem of *classification* of graph streams. Along this direction, Chi et al. [8] proposed a fast graph stream classification algorithm that uses discriminative clique hashing (DICH), which can be applicable for OLAP analysis over evolving complex networks. Furthermore, Valari et al. [27] discovered top- $k$  dense subgraphs in dynamic graph collections by means of both exact and approximate algorithms. As a preview, while these recent studies focus on graph mining, the mining algorithms we propose in the current paper work on both graph-structured data and other non-graph data.

Note that, although memory is not too expensive nowadays, the volume of data generated in data streams (including graph streams) also keeps growing at a rapid rate. Hence, algorithms for mining frequent patterns with limited memory are still in demand, so as to deal with the probing case of streams generated by graph data sources. For instance, Cameron et al. [6] studied this topic and proposed an algorithm that works well for *sparse* data streams in limited memory space. In contrast, the mining algorithms we propose in the current paper are designed to mine *dense* data streams in limited memory space. These algorithms can be viewed as complements to the sparse stream mining algorithm.

Here, *key contributions* of our current paper include a simple yet powerful on-disk data structure called *DSMatrix* for capturing and maintaining relevant data found in the streams, including dense graph streams. The DSMatrix is designed for *stream mining of frequent patterns* with window-sliding models. The corresponding tree-based *hori-*

*zontal mining algorithm* builds a tree for data in the current window captured in the DSMatrix. Moreover, our *frequency counting technique* effectively avoids the recursive building of FP-trees for projected databases, and thus saving space. Furthermore, our *vertical mining algorithm* takes advantage of the data representation of the DSMatrix to mine frequent patterns efficiently. As the proposed DSMatrix can generally be applicable to different kinds of streaming data, it can be used in graph streams where memory requirements are very demanding.

This paper is organized as follows. Background is provided in Section 2. Section 3 presents our DSMatrix structure for capturing important information from dense streams and describes how our DSMatrix can efficiently mine frequent patterns from graph streams. Then, we discuss how we make use of the DSMatrix for horizontal (Section 4) and vertical (Section 5) mining of frequent patterns extracted from graph streams. Section 6 focuses on an analytical evaluation of the properties of the DSMatrix structure in comparison with other similar (stream) frequent pattern structures. Section 7 shows experimental results. Finally, conclusions are given in Section 8.

## 2. BACKGROUND

To mine frequent patterns, an exact stream mining algorithm [21] first constructs a **Data Stream Tree (DSTree)**, which is then used as a *global tree* for recursively generating smaller FP-trees (as *local trees*) for projected databases. Due to the dynamic nature of data streams (as seen in Properties 1 and 2), frequencies of items are continuously affected by the insertion of new batches (and the removal of old batches) of transactions. Arranging items in frequency-dependent order may lead to swapping—which, in turn, can cause merging and splitting—of tree nodes when frequencies change. Hence, in the DSTree, transaction items are arranged according to some canonical order (e.g., alphabetical order), which can be specified by the user prior to the tree construction or mining process. Consequently, the DSTree can be constructed using only a *single* scan of the graph streams. Note that the DSTree is designed for processing streams within a sliding window. So, for a window size of  $w$  batches, each tree node keeps (i) an item and (ii) a *list* of  $w$  frequency values (instead of a single frequency count in each node of the FP-tree for frequent pattern mining from *static* databases). Each entry in this list captures the frequency of the item in each batch of dynamic streams in the current window. By so doing, when the window slides (i.e., when new batches are inserted and old batches are deleted), frequency information can be updated easily. Consequently, the resulting DSTree preserves the usual tree properties that (i) the total frequency (i.e., sum of  $w$  frequency values) of any node is at least as high as the sum of total frequencies of its children and (ii) the ordering of items is unaffected by the continuous changes in item frequencies.

With the aforementioned DSTree, the mining is “delayed” until it is needed. Hence, once the DSTree is constructed, it is always kept up-to-date when the window slides. The exact mining algorithm mines frequent patterns from the updated DSTree by performing the following steps. It first traverses/extracts relevant tree paths upwards and sums the frequency values of each list in a node representing an item (or itemset)—to obtain its frequency in the current sliding window—for forming an appropriate projected database.



Afterwards, the algorithm constructs a FP-tree for the projected database of each of these frequent patterns of only 1 item (i.e., 1-itemset) such as an  $\{x\}$ -projected database (in a similar fashion as in the FP-growth algorithm for mining static data [16]). Thereafter, the algorithm recursively forms subsequent FP-trees for projected databases of frequent  $k$ -itemsets where  $k \geq 2$  (e.g.,  $\{x, y\}$ -projected database,  $\{x, z\}$ -projected database, etc.) by traversing paths in these FP-trees. As a result, the algorithm finds all frequent patterns. Note that, as items are consistently arranged according to some canonical order, the algorithm guarantees the inclusion of all *frequent* items using just upward traversals. Moreover, there is also no worry about possible omission or double-counting of items during the mining process. Furthermore, as the DSTree is always kept up-to-date, all frequent patterns—which are embedded in batches within the current sliding window—can be found effectively.

In the remainder of this paper, we call the aforementioned exact algorithm that uses the DSTree as the global tree, from which FP-trees for subsequent projected databases can be constructed recursively, the **(global DSTree, local FP-trees) mining option**. It works well when memory space is not an issue. The success of this algorithm mainly relies on the assumption—usually made for many tree-based algorithms [16]—that all tree (i.e., the global tree together with subsequent FP-trees) fit into the memory. For example, when mining frequent patterns from the  $\{x, y, z\}$ -projected database, the global tree and two subsequent FP-trees (for the  $\{x\}$ -,  $\{x, y\}$ - and  $\{x, y, z\}$ -projected databases) are all assumed to be fit into memory.

However, there are situations where the memory is so limited that not all the trees can fit into memory, like the case of streaming generated from graph data sources. To handle these situations, the **Data Stream Table (DSTable)** [6] was proposed. The DSTable is a two-dimensional table that captures on the disk the contents of transactions in all batches within the current sliding window. Each row of the DSTable represents a domain item. Like the DSTree, items in the DSTable are arranged according to some canonical order (e.g., alphabetical order), which can be specified by the user prior to the construction of the DSTable. As such, table construction requires only a single scan of the graph stream. Each entry in the resulting DSTable is a “pointer” that points to the location of the table entry (i.e., which row and which column) for the “next” item in the same transaction. When dealing with graph streaming data, the DSTable also keeps  $w$  boundary values (to represent the boundary between  $w$  batches in the current sliding window) for each item. By doing so, when the window slides, transactions in the old batch can be removed and transactions in the new batch can be added easily.

Similar to mining with the DSTree, the mining with this DSTable is also “delayed” until it is needed. Once the DSTable is constructed and kept up-to-date when the window slides, the mining algorithm first traverses/extracts relevant transactions from the DSTable. Then, the algorithm (i) constructs a FP-tree for the projected database of each of these 1-itemsets and (ii) recursively forms subsequent FP-trees for projected databases of frequent  $k$ -itemsets (where  $k \geq 2$ ) by traversing the paths of these FP-trees. As a result, the algorithm finds all frequent patterns. In the remainder of this paper, we call this the **(global DSTable, local FP-trees) mining option**.

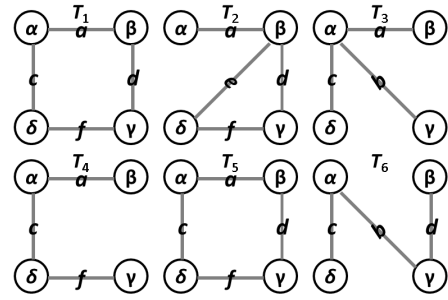


Figure 1: A graph stream (Example 1).

*Example 1.* For illustrative purpose, let us consider a sliding window of size  $w = 2$  batches (i.e., only two batches are kept) and the following segments in a stream of graphs, where each graph  $G = (V, E)$  consists of  $|V| = 4$  vertices (Vertices  $\alpha, \beta, \gamma$  and  $\delta$ ) and  $|E| \leq 6$  edges:

- At time  $T_1$ ,  $E = \{(\alpha, \beta), (\alpha, \delta), (\beta, \gamma), (\gamma, \delta)\}$ ;
- At time  $T_2$ ,  $E = \{(\alpha, \beta), (\beta, \gamma), (\beta, \delta), (\gamma, \delta)\}$ ;
- At time  $T_3$ ,  $E = \{(\alpha, \beta), (\alpha, \gamma), (\alpha, \delta)\}$ ;
- At time  $T_4$ ,  $E = \{(\alpha, \beta), (\alpha, \delta), (\gamma, \delta)\}$ ;
- At time  $T_5$ ,  $E = \{(\alpha, \beta), (\alpha, \delta), (\beta, \gamma), (\gamma, \delta)\}$ ; and
- At time  $T_6$ ,  $E = \{(\alpha, \gamma), (\alpha, \delta), (\beta, \gamma)\}$ .

See Figure 1. These graphs may represent some interactions in meetings or friendships among social individuals. For simplicity, we represent these edges by six symbols  $a, b, c, d, e$  and  $f$ . Consequently, we get (i) transactions  $t_1 = \{a, c, d, f\}$ ,  $t_2 = \{a, d, e, f\}$  and  $t_3 = \{a, b, c\}$  in the first batch  $B_1$ ; as well as (ii) transactions  $t_4 = \{a, c, f\}$ ,  $t_5 = \{a, c, d, f\}$  and  $t_6 = \{b, c, d\}$  in the second batch  $B_2$ . Let the user-specified *minsup* threshold be 2. Then, the DSTable stores the following information:

DSTable:

ROW	BOUNDARIES	CONTENTS
Edge $a$ :	Cols 3 & 5	$(c, 1), (d, 2), (b, 1); (c, 3), (c, 4)$
Edge $b$ :	Cols 1 & 2	$(c, 2); (c, 5)$
Edge $c$ :	Cols 2 & 5	$(d, 1), \text{end}; (f, 3), (d, 3), (d, 4)$
Edge $d$ :	Cols 2 & 4	$(f, 1), (e, 1); (f, 4), \text{end}$
Edge $e$ :	Cols 1 & 1	$(f, 2);$
Edge $f$ :	Cols 2 & 4	$\text{end}, \text{end}; \text{end}, \text{end}$

In the DSTree, the first entry in Row  $a$  with value  $(c, 1)$ —which captures a transaction starting edge/item  $a$  and having  $c$  as the second edge—points to the 1st column of Row  $c$ . Its value  $(d, 1)$  points to the 1st column of Row  $d$ , which captures the value  $(f, 1)$ . This indicates the third and fourth edges are  $d$  and  $f$ , respectively. Then, the 1st column of Row  $e$  with value “end” indicates the end of the transaction containing  $\{a, c, d, f\}$ . Based on contents of the entire DSTable, the mining algorithm first finds frequent singletons  $\{a\}, \{b\}, \{c\}, \{d\}$  and  $\{f\}$ . The algorithm then constructs an FP-tree for the  $\{a\}$ -projected database (i.e., transactions containing  $a$ ) to get frequent 2-itemsets  $\{a, c\}, \{a, d\}$  and  $\{a, f\}$ . From this FP-tree, the algorithm recursively constructs subsequent FP-trees (e.g., for  $\{a, c\}$ -,  $\{a, c, d\}$ - and  $\{a, d\}$ -projected databases). Afterwards, the algorithm constructs an FP-tree for the  $\{b\}$ -projected database (i.e., transactions containing  $b$ ), from which subsequent FP-trees are constructed. Similar steps apply to  $\{c\}$ - and  $\{d\}$ -projected databases.

The boundary information “Cols 3 & 5” for Row  $a$  indicates that (i) the boundary between batches  $B_1$  and  $B_2$  is at the end of column 3 and (ii) batch  $B_2$  ends at column 5. Hence, when a new batch comes in, the old batch is removed. In this case, the first three columns of Row  $a$  (due to “Cols 3 & 5” in Row  $a$ ), the first 1 column of Row  $b$  (due to “Cols 1 & 2” in Row  $b$ ), the first 2 columns of Rows  $c$  and  $d$ , the first 1 column of Row  $e$ , as well as the first 2 columns of Row  $f$  can be removed.  $\square$

Observed from the above example, mining with the (global DSTree, local FP-trees) option may suffer from several problems when handling data streams (especially, dense graph streams) with limited memory. Some of these problems are listed as follows:

- P1. To facilitate easy insertion and deletion of contents in the DSTable when the window (of size  $w$  batches) slides, the DSTable keeps  $w$  boundary values for each row (representing each of the  $m$  domain items). Hence, the DSTable needs to keep a total of  $m \times w$  boundary values.
- P2. Each table entry is a “pointer” that indicates the location in terms of row name (e.g., Row  $c$ ) and column number (e.g., Column 1) of the table entry for the “next” item in the same transaction. When the data stream is sparse, only a few “pointers” need to be stored. However, when the graph stream is dense, many “pointers” need to be stored. Given a total of  $|T|$  transactions in all batches within the current sliding window, there are potentially  $m \times |T|$  “pointers” (where  $m$  is the number of domain items).
- P3. During the mining process, multiple FP-trees need to be constructed and kept in memory (e.g., FP-trees for all  $\{a\}$ -,  $\{a, c\}$ - and  $\{a, c, d\}$ -projected databases are required to be kept in memory).

### 3. THE DSMatrix DATA STRUCTURE

In attempt to solve the above problems while mining frequent patterns from data streams (especially, dense graph streams) with limited memory, we propose a 2-dimensional structure called **Data Stream Matrix (DSMatrix)**. This matrix structure captures the contents of transactions in all batches within the current sliding window by storing them on the disk. Note that the DSMatrix is a binary matrix, which represents the presence of an item  $x$  in transaction  $t_i$  by a “1” in the matrix entry  $(t_i, x)$  and the absence of an item  $y$  from transaction  $t_j$  by a “0” in the matrix entry  $(t_j, y)$ . With this binary representation of items in each transaction, each column in the DSMatrix captures a transaction. Each column in the DSMatrix can be considered as a bit vector.

Similar to the DSTable, our DSMatrix also keeps track of any boundary between two batches so that, when the window slides, transactions in the older batches can be easily removed and transactions in the newer batches can be easily added. Note that, in the DSTable, boundaries may vary from one row (representing an item) to another row (representing another item) due to the potentially different number of items present. Contrarily, in our DSMatrix, boundaries are the same from one row to another because we put a binary value (0 or 1) for each transaction.

*Example 2.* Let us revisit Example 1. The information captured by that DSTable can be effectively captured by our DSMatrix, but in less space:

**Our DSMatrix:**

BOUNDARIES: Cols 3 & 6	
ROW	CONTENTS
Row $a$ :	1 1 1; 1 1 0
Row $b$ :	0 0 1; 0 0 1
Row $c$ :	1 0 1; 1 1 1
Row $d$ :	1 1 0; 0 1 1
Row $e$ :	0 1 0; 0 0 0
Row $f$ :	1 1 0; 1 1 0

When compared with the DSTable, we do not need to store the same boundary information multiple times (for the  $m$  domain items). We only need to store it once.  $\square$

Hence, with our DSMatrix, we solve previous Problems P1 and P2 of the DSTree, as follows:

- S1. Recall that the DSTable needs to keep a total  $m \times w$  boundary values. In contrast, our DSMatrix only keeps  $w$  boundary values (where  $w \ll m \times w$ ) for the entire matrix, regardless how many domain items ( $m$ ) are here.
- S2. Recall that each table entry in the DSTable captures both the row name and column number to represent a “pointer” to the next item in a transaction. The computation of column number requires the DSTable to constantly keep track of the index of the last item in each row representing a domain item. Moreover, each “pointer” requires two integer (row name/number and column number). For  $P$  items in  $|T|$  transactions, the DSTable requires  $2 \times 32 \times P$  bits (for 32-bit integer representation). For dense data streams, the DSTable requires potentially  $64m \times |T|$  bits. In contrast, our DSMatrix uses a bit vector to indicate the presence or absence of items in a transaction. The computation does not require us to keep track of the index of the last item in every row and thus incurring a lower computation cost. Moreover, given a total of  $|T|$  transactions in all batches within the current sliding window, there are  $|T|$  columns in our DSMatrix. Each column requires only  $m$  bits. In other words, our DSMatrix takes  $m \times |T|$  bits (cf. potentially  $64m \times |T|$  bits for dense data streams required by the DSTree).

### 4. TREE-BASED HORIZONTAL FREQUENT PATTERN MINING

Whenever a new batch of streaming data (e.g., streaming graph data) comes in, the window slides. Transactions in the oldest batch in the sliding window are then removed from our DSMatrix so that transactions in this new batch can be added. Following the aforementioned mining routines, the mining is “delayed” until it is needed. Once the DSMatrix is constructed, it is kept up-to-date on the disk.

To find frequent patterns, we propose a *tree-based horizontal mining algorithm*. When the user needs to find frequent patterns, we extract relevant transactions from the DSMatrix to form an FP-tree for each projected database of every frequent singleton. Key ideas of the algorithm are illustrated in Example 3.

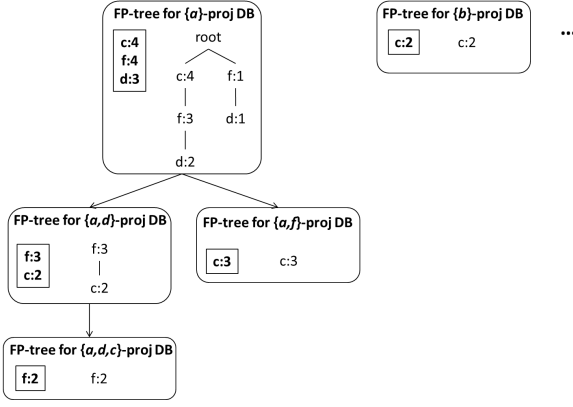


Figure 2: Multiple FP-trees built for  $\{a\}$ -,  $\{b\}$ -, ... projected DBs from the DSMatrix (Example 3).

*Example 3.* Continue with Example 2. To form the  $\{a\}$ -projected database, we examine Row  $a$ . For every column with a value “1”, we extract its column downwards (e.g., from edges/items  $b$  to  $e$  if they exist). Specifically, when examining Row  $a$ , we notice that columns 1, 2, 3, 4 and 6 contain values “1” (which means that  $a$  appears in those five transactions in the two batches of streaming graph data in the current sliding window). Then, from Column 1, we extract  $\{c, d, f\}$ . Similarly, we extract  $\{d, e, f\}$  and  $\{b, c\}$  from Columns 2 and 3. We also extract  $\{c, f\}$  and  $\{c, d, f\}$  from columns 4 and 5. All these form the  $\{a\}$ -projected database, from which an FP-tree can be built. From this FP-tree for the  $\{a\}$ -projected database, we find that 2-itemsets  $\{a, c\}$ ,  $\{a, d\}$  and  $\{a, f\}$  are frequent. Hence, we then form  $\{a, d\}$ - and  $\{a, f\}$ -projected databases, from which FP-trees can be built. (Note that we do not need to form the  $\{a, c\}$ -projected database as it is empty after forming both  $\{a, d\}$ - and  $\{a, e\}$ -projected databases.) When applying this step recursively in a depth-first manner, we obtain frequent 3-itemsets  $\{a, c, d\}$ ,  $\{a, c, f\}$  and  $\{a, d, f\}$ , which leads to FP-trees for the  $\{a, d, c\}$ -projected database. (Again, we do not need to form the  $\{a, f, c\}$ - or  $\{a, d, f\}$ -projected databases as they are both empty.) At this moment, we keep FP-trees for the  $\{a\}$ -,  $\{a, d\}$ - and  $\{a, d, c\}$ -projected databases. Afterwards, we also find that 4-itemset  $\{a, c, d, f\}$  is frequent. In the context of graph streams, this is a frequent collection of 4 edges—namely, Edges  $a, c, d$  and  $f$ . See Figure 2.

We backtrack and examine the next frequent singleton  $\{b\}$ . When examining Row  $b$ , we notice that Columns 3 and 6 contain values “1” (which means that  $b$  appears in those two transactions in the current sliding window). For these two columns, we extract downward to get  $\{c\}$  and  $\{c, d\}$  that appear together with  $b$  (i.e., to form the  $\{b\}$ -projected database. As shown in Figure 2, the corresponding FP-tree contains  $\{c\}:2$  meaning that  $c$  occurs twice with  $b$  (i.e., 2-itemset  $\{b, c\}$  is frequent with frequency 2). Similar steps are applied to other frequent singletons  $\{c\}$ ,  $\{d\}$  and  $\{f\}$  in order to discover all frequent patterns.  $\square$

Note that, during the mining process, we require multiple FP-trees to be kept in the memory during the mining process (i.e., Problem P3 of the (global DSTree, local FP-trees) mining option). However, when the memory space is limited, *not* all of the multiple FP-trees can fit into the memory.

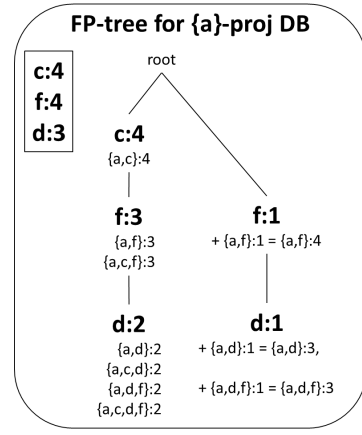


Figure 3: FP-tree for  $\{a\}$ -proj. DB (Example 4).

To solve this problem, which identifies the Problem P3 above, we propose the following effective frequency counting technique:

- S3. Once an FP-tree for the projected database of a frequent singleton is built, we traverse every tree node in a depth-first manner (e.g., pre-order, in-order, or post-order traversal). For every first visit of a tree node, we generate the itemset represented by the node and its subsets. We also compute their frequencies.

*Example 4.* Based on the DSMatrix in Example 2, we first construct an FP-tree for the  $\{a\}$ -projected database. Then, we traverse every node in such an FP-tree. When traversing the leftmost branch  $\langle c:4, b:1 \rangle$ , we visit nodes “ $c:4$ ” (which represents itemset  $\{a, c\}$  with frequency 4) and “ $b:1$ ” (which gives  $\{a, b\}$  with frequency 1 and  $\{a, b, c\}$  with frequency 1). Next, we traverse the middle branch  $\langle c:4, f:3, d:2 \rangle$ . By visiting nodes “ $f:3$ ” and “ $d:2$ ”, we get  $\{a, f\}$  and  $\{a, c, f\}$  both with frequencies 3, as well as  $\{a, d\}$ ,  $\{a, c, d\}$ ,  $\{a, d, f\}$  and  $\{a, c, d, f\}$  all with frequencies 2. Finally, we visit nodes “ $f:1$ ” and “ $d:1$ ” in the rightmost branch  $\langle f:1, d:1 \rangle$ , from which we get the frequency 1 for both  $\{a, d\}$ ,  $\{a, d, f\}$  and  $\{a, f\}$ . This frequency value is added to the existing frequency count of 2 (from the middle branch) to give the frequency of  $\{a, d\}$  and  $\{a, d, f\}$  equal to 3. Hence, with the *minsup* threshold set to 2, we obtain frequent patterns  $\{a, c\}:4$ ,  $\{a, c, d\}:2$ ,  $\{a, c, d, f\}:2$ ,  $\{a, c, f\}:3$ ,  $\{a, d\}:3$ ,  $\{a, d, f\}:3$  and  $\{a, f\}:4$ . Note that, during this mining process for the  $\{a\}$ -projected database, we count frequencies of itemsets without recursive construction of FP-trees. See Figure 3.

Afterwards, we build an FP-tree for the  $\{b\}$ -projected database and count frequencies of all frequent patterns containing item  $b$ . Similar steps are applied to the FP-trees for the  $\{c\}$ - and  $\{d\}$ -projected databases.  $\square$

Note that, at any moment during the mining process, only one FP-tree needs to be constructed and kept in the memory for this (global DSMatrix, local FP-tree) mining process (cf. multiple FP-trees required for the (global DSTree, multiple local FP-trees) mining option). This solves Problem P3.

## 5. VERTICAL FREQUENT PATTERN MINING

In Section 4, we described a tree-based horizontal frequent pattern mining algorithm that makes good use of the DSMatrix to form FP-trees for frequent singletons. From each of these FP-trees, the algorithm applies an effective frequency counting technique to find all frequent patterns with their frequency information in such a way that the algorithm avoids recursive construction of FP-trees for frequent  $k$ -itemsets (where  $k > 2$ ).

In this section, we present a *vertical frequent pattern mining algorithm*. Given that the contents stored in our DSMatrix can be considered as a collection of bit vectors. It becomes logical to consider vertical mining. To mine frequent singletons, we examine each row (representing a domain item). The *row sum* (i.e., total number of 1s) gives the frequency of the item represented by that row. Once the frequent singletons are found, we *intersect the bit vectors* for two items. If the row sum of the resulting intersection  $\geq$  the user-specified *minsup* threshold, then we find a frequent 2-itemset. We repeat these steps by intersecting two bit vectors of frequent patterns to find frequent patterns of higher cardinality.

*Example 5.* Based on the DSMatrix in Example 1, we first compute the row sum for each row (i.e., for each domain item). As a result, we find that edges/items  $a, b, c, d$  and  $f$  are all frequent with frequencies 5, 2, 5, 4 and 4, respectively. Afterwards, we intersect the bit vector of  $a$  (i.e., Row  $a$ ) with any one of the remaining four bit vectors (i.e., any one of the four rows) to find frequent 2-itemsets  $\{a, c\}$ ,  $\{a, d\}$  and  $\{a, f\}$  with frequencies 4, 3 and 4, respectively, because (i) the intersection of  $\vec{a}$  and  $\vec{c}$  gives a bit vector 101110, (ii) the intersection of  $\vec{a}$  and  $\vec{d}$  gives a bit vector 110010, and (iii) the intersection of  $\vec{a}$  and  $\vec{f}$  gives a bit vector 110110. Next, we intersect (i)  $\vec{ac}$  with  $\vec{ad}$ , (ii)  $\vec{ac}$  with  $\vec{af}$  and (iii)  $\vec{ad}$  with  $\vec{af}$  to find frequent 3-itemsets  $\{a, c, d\}$ ,  $\{a, c, f\}$  and  $\{a, d, f\}$ . We also intersect  $\vec{acd}$  with  $\vec{acf}$  to find frequent 4-itemset  $\{a, c, d, f\}$ . These are all frequent patterns containing item  $a$ .

Afterwards, we repeat similar steps with the bit vectors for the other singletons. For instance, we intersect  $\vec{b}$  with  $\vec{c}$ ,  $\vec{d}$  and  $\vec{f}$ . We find out that, among them, only  $\{b, c\}$  is frequent with frequency 2. We also intersect  $\vec{c}$  with  $\vec{d}$  and  $\vec{f}$  to find frequent 3-itemsets  $\{c, d\}$  and  $\{c, f\}$ , each with frequencies of 3. We also find frequent 4-itemsets  $\{c, d, f\}$  by intersecting  $\vec{cd}$  and  $\vec{cf}$ . Finally, we intersect  $\vec{d}$  and  $\vec{f}$  to find frequent 2-itemset  $\{d, f\}$  with frequency 3.  $\square$

## 6. ANALYTICAL EVALUATION

Recall from Section 1 that FP-streaming [14] is an approximate algorithm, which uses an “immediate” mode for mining. During the mining process for each batch of the streaming data, FP-streaming builds a global FP-tree and  $O(f \times d)$  subsequent local FP-trees, where  $f$  is the number of frequent items in the domain and  $d$  is the height/depth of the global FP-tree. At any time during the mining process for each batch, the global FP-tree and  $O(d)$  subsequent local FP-trees are stored in the memory. Hence, storage cost includes the memory space for the global FP-tree plus all  $O(d)$  subsequent local FP-trees for each batch. In terms of efficiency, when the number of batches in the data stream increases, the CPU cost for the mining process increases. For

example, let us consider a sliding window of  $w=5$  batches. When handling a stream of  $S=100$  batches, FP-streaming builds the global FP-tree plus all subsequent local FP-trees, and mines frequent patterns from these FP-trees for each of the 100 batches. In other words, FP-streaming builds 100 sets of the global and subsequent local FP-trees. Moreover, the computation effort for the first 95 batches is wasted (when users request frequent patterns at the end of the 100th batch).

Recall from Section 2 that mining with the DSTree [21] or DSTable [6] uses a “delayed” mode for mining. So, the actual mining of frequent patterns is delayed until they are needed to be returned to the user. Hence, for  $S=100$  batches, the mining algorithm needs to build a global DSTree or DSTable and updates it  $S-w = 100-5 = 95$  times. Once an updated DSTree or DSTable has captured the 96th to the 100th batches, multiple FP-trees are constructed to find frequent patterns. Note that only one set of the updated global DSTree (or DSTable) and multiple FP-trees are required (cf. building 100 sets of a global tree and  $O(f \times d)$  FP-trees by FP-streaming, one set for each of the 100 batches). Moreover, at any time during the mining process of the (global DSTree, local FP-trees) option, only the global DSTree and multiple FP-trees are needed to be present (cf. one global FP-tree and  $O(d)$  subsequent FP-trees are needed to be present in FP-streaming). When using the (global DSTable, local FP-trees) option, the global DSTable is kept on disk. Thus, only multiple FP-trees are needed to be kept in the memory.

In contrast, the DSMatrix resides on disk. Being specialized to dense graph streams, it can serve as an alternative to the global FP-tree when memory is limited. Moreover, the size of the DSMatrix is independent of the user-specified minimum support threshold (*minsup*). Hence, it is useful for interactive mining, especially when users keep adjusting *minsup*, which is relevant for mining graph streams. It should be noted that the DSMatrix captures the transactions in the current sliding window. During the mining process, the algorithm skips infrequent items (i.e., items having support lower than *minsup*) and only includes frequent items when building subsequent FP-trees for projected databases. Furthermore, with our frequency counting technique, we do not even need to build too many FP-trees. Instead, we only need to build FP-trees for frequent singleton (i.e., for  $\{x\}$ -projected databases, where  $x$  is a frequent item). When users adjust *minsup* during the interactive mining process, we do not need to rebuild the DSMatrix. In contrast, when *minsup* changes, FP-streaming needs to rebuild the global FP-tree.

In terms of disk space, the DSTable [6] requires  $64 \times P$  bits (for 32-bit integer representation), where  $P$  is the total number of items in  $|T|$  transactions in the  $w$  batches of the data streams. In the worst case, the DSTable requires potentially  $64m \times |T|$  bits for dense data streams. In contrast, our DSMatrix requires only  $m \times |T|$  bits, which is desirable for applications that require dense graph stream mining.

## 7. EXPERIMENTAL EVALUATION

To acquire dense graph stream datasets, we first generated random graph models via a Java-based generator by varying model parameters (e.g., topology, average fan-out of nodes, edge centrality, etc.). We then generated graph streams as nodes and node-edge relationships derived from the above

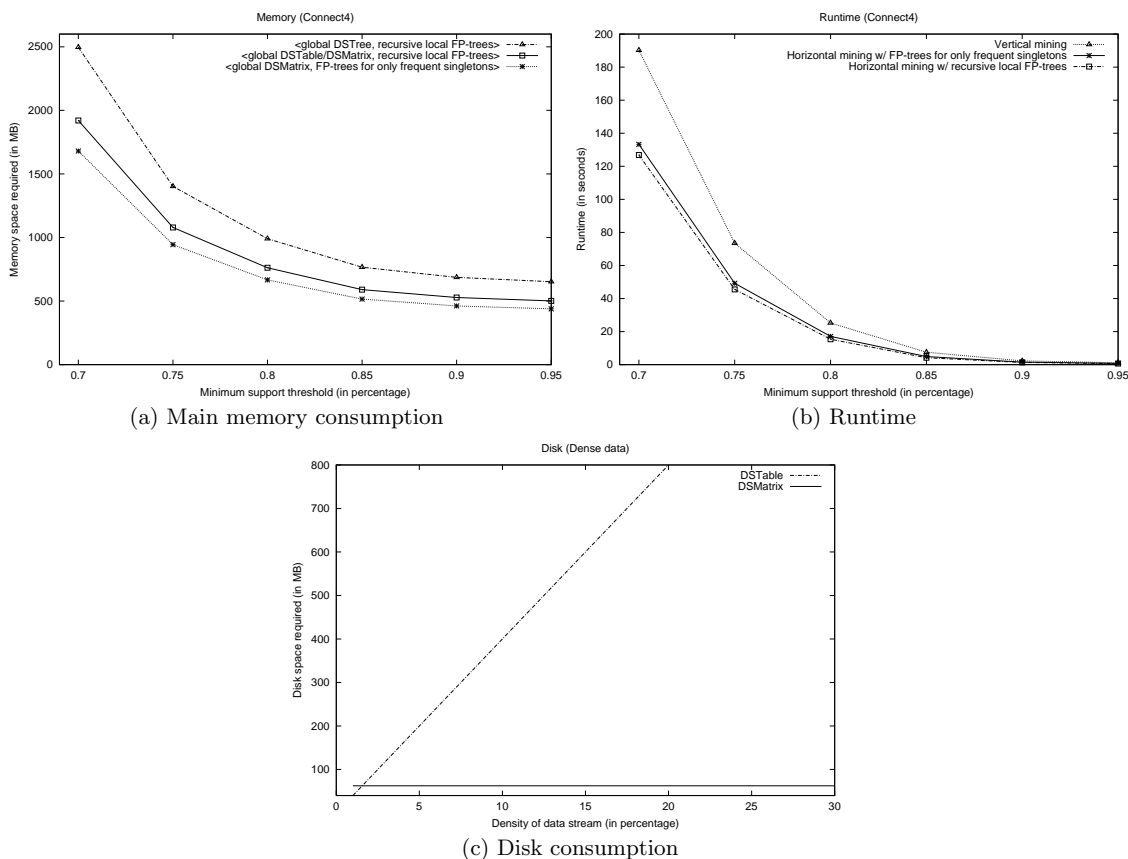


Figure 4: Experimental results.

graph models, and obtained node values from popular data stream sets available in literature (stored in the projected database). In addition, we also used many different databases including IBM synthetic data, real-life databases (e.g., connect4) from the UC Irvine Machine Learning Depository as well as those from the Frequent Itemset Mining Implementation (FIMI) Dataset Repository. For example, connect4 is a dense data set containing 67,557 records with an average transaction length of 43 items, and a domain of 130 items. Each record represents a graph of legal 8-ply positions in the game of connect 4.

All experiments were run in a time-sharing environment in a 1 GHz machine. We set each batch to be 6K records and the window size  $w=5$  batches. The reported figures are based on the average of multiple runs. Runtime includes CPU and I/Os; it includes the time for both tree construction and frequent pattern mining steps. In the experiments, we mainly evaluated the accuracy and efficiency of our DMatrix by comparing with related works such as (i) DSTree [21] and (ii) DSTable [6].

In the first experiment, we measured the accuracy of the following four mining options: (i) <global DSTree, recursive local FP-trees>; (ii) <global DSTable, recursive local FP-trees>; (iii) <global DMatrix, recursive local FP-trees>; (iv) <global DMatrix, local FP-trees for only frequent singletons> options. Experimental results show that mining with any of these four options give the same mining results.

While these four options gave the same results, their performance varied. In the second and third experiments, we measured the space and time efficiency of our proposed DMatrix. Results show that the <DSTree, recursive FP-trees> option required the largest main memory space as it stores one global DSTree and multiple local FP-trees in main memory. The <DSTable, recursive FP-trees> and <DMatrix, recursive FP-trees> options required less memory as their DSTable and DMatrix were kept on disk. Among the four mining options, the <DMatrix, FP-trees for only frequent singletons> option required the *smallest* main memory space because at most  $m$  FP-trees needed to be generated during the entire mining process, one for each frequent domain item. Note that not all  $m$  domain items were frequent. Moreover, at any mining moment, only one of these FP-trees needs to be presented. In other words, not all  $\leq m$  FP-trees were generated at the same time. See Figure 4(a).

There are tradeoffs between memory consumption vs. runtime performance. Runtime performance of the three options also varied. For instance, vertical mining does not consume too much memory. While the bitwise operation (e.g., intersection) is quick, but vertical mining required many scans to read bit vectors from the DMatrix. In contrast, horizontal mining with FP-trees built for only frequent singletons consumes more memory because it keeps these  $\leq m$  FP-trees from  $\leq m$  frequent singletons in memory. Along the same direction, horizontal mining with FP-

trees recursively built for subsequent frequent patterns consumes even more memory because it keeps all FP-trees (i.e., those for frequent singletons and their subsequent frequent  $k$ -itemsets, where  $k \geq 2$ ) in memory. See Figure 4(a). It is important to note that reading from disk would be a *logical choice* in a limited main memory environment.

Moreover, we perform some additional experiments, by testing with the usual experiment (e.g., the effect of *minsup*). As shown in Figure 4(b), the runtime decreased when *minsup* increased. In another experiment, we tested scalability with the number of transactions. The results show that mining with our proposed DSMatrix was scalable (see Figure 4(c)). In particular, Figure 4(c) compares the disk consumption between the DSTable and our DSMatrix, and it clearly shows that our DSMatrix requires a constant amount of disk space, where the DSTable requires different amounts depending on the density of data streams. An interesting observation that, for dense data, our DSMatrix is beneficial due to its bit vector representation. As future work, we plan to conduct more extensive experiments on various datasets (including Big data) with different parameter settings (e.g., varying *minsup* and transaction lengths that represent the complexity of graphs).

## 8. CONCLUSIONS

As technology advances, streams of data (including graph streams) can be produced in many applications. Key contributions of this paper include (i) a simple yet powerful alternative disk-based structure—called *DSMatrix*—for efficient frequent pattern mining from streams (e.g., dense graph streams) with limited memory and (ii) two frequent pattern mining algorithms: a *tree-based horizontal mining algorithm* and a *vertical mining algorithm*. To avoid keeping too many FP-trees in memory when the space is limited, we also described an effective frequency counting technique, which requires only one FP-tree for a projected database to be kept in the limited memory. Analytical and experimental results show the benefits of our DSMatrix structure and its corresponding mining algorithms.

## 9. ACKNOWLEDGEMENTS

This project is partially supported by NSERC (Canada) and University of Manitoba.

## 10. REFERENCES

- [1] C.C. Aggarwal. On classification of graph streams. In *Proc. SDM 2011*, pp. 652–663.
- [2] C.C. Aggarwal, Y. Li, P.S. Yu, & R. Jin. On dense pattern mining in graph streams. *PVLDB*, **3**(1–2), pp. 975–984 (2010).
- [3] R. Agrawal & R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB 1994*, pp. 487–499.
- [4] A. Bifet, G. Holmes, B. Pfahringer, & R. Gavaldà. Mining frequent closed graphs on evolving data streams. In *Proc. ACM KDD 2011*, pp. 591–599.
- [5] G. Buehrer, S. Parthasarathy, & A. Ghoting. Out-of-core frequent pattern mining on a commodity. In *Proc. ACM KDD 2006*, pp. 86–95.
- [6] J.J. Cameron, A. Cuzzocrea, & C.K. Leung. Stream mining of frequent sets with limited memory. In *Proc. ACM SAC 2013*, pp. 173–175.
- [7] J.J. Cameron, C.K. Leung, & S.K. Tanbeer. Finding strong groups of friends among friends in social networks. In *Proc. SCA 2011*, pp. 824–831.
- [8] L. Chi, B. Li, & X. Zhu. Fast graph stream classification using discriminative clique hashing. In *Proc. PAKDD 2013, Part I*, pp. 225–236.
- [9] D.Y. Chiu, Y.H. Wu, & A. Chen. Efficient frequent sequence mining by a dynamic strategy switching algorithm. *VLDBJ*, **18**(1), pp. 303–327 (2009).
- [10] A. Cuzzocrea. CAMS: OLAPing Multidimensional Data Streams Efficiently. In *Proc. DaWaK 2009*, pp. 48–62.
- [11] A. Cuzzocrea & S. Chakravarthy. Event-based lossy compression for effective and efficient OLAP over data streams. *Data & Knowledge Engineering*, **69**(7), pp. 678–708 (2010).
- [12] A. Cuzzocrea, F. Furfaro, G.M. Mazzeo & D. Saccà. A Grid Framework for Approximate Aggregate Query Answering on Summarized Sensor Network Readings. In *Proc. OTM Workshops 2004*, pp. 144–153.
- [13] A. Fariha, C.F. Ahmed, C.K. Leung, S.M. Abdullah, & L. Cao. Mining frequent patterns from human interactions in meetings using directed acyclic graphs. In *Proc. PAKDD 2013, Part I*, pp. 38–49.
- [14] C. Giannella, J. Han, J. Pei, X. Yan, & P.S. Yu. Mining frequent patterns in data streams at multiple time granularities. In *Data Mining: Next Generation Challenges and Future Directions*, ch. 6 (2004).
- [15] G. Grahne & J. Zhu. Mining frequent itemsets from secondary memory. In *Proc. IEEE ICDM 2004*, pp. 91–98.
- [16] J. Han, J. Pei, & Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD 2000*, pp. 1–12.
- [17] R. Jin & G. Agrawal. An algorithm for in-core frequent itemset mining on streaming data. In *Proc. IEEE ICDM 2005*, pp. 210–217.
- [18] C.K. Leung. Mining frequent itemsets from probabilistic datasets. In *Proc. EDB 2013*, pp. 137–148.
- [19] C.K. Leung & C.L. Carmichael. Exploring social networks: a frequent pattern visualization approach. In *Proc. IEEE SocialCom 2010*, pp. 419–424.
- [20] C.K. Leung, A. Cuzzocrea, & F. Jiang. Discovering frequent patterns from uncertain data streams with time-fading and landmark models. *LNCS TLDKS*, **8**, pp. 174–196 (2013).
- [21] C.K. Leung & Q.I. Khan. DSTree: a tree structure for the mining of frequent sets from data streams. In *Proc. IEEE ICDM 2006*, pp. 928–932.
- [22] C.K. Leung & S.K. Tanbeer. PUF-tree: a compact tree structure for frequent pattern mining of uncertain data. In *Proc. PAKDD 2013, Part I*, pp. 13–25.
- [23] F. Mandreoli, R. Martoglia, G. Villani, & W. Penzo. Flexible query answering on graph-modeled data. In *Proc. EDBT 2009*, pp. 216–227.
- [24] O. Papapetrou, M. Garofalakis, & A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *PVLDB*, **5**(10), pp. 992–1003 (2012).
- [25] S.K. Tanbeer, F. Jiang, C.K. Leung, R.K. MacKinnon, & I.J.M. Medina. Finding groups of friends who are significant across multiple domains in social networks. In *Proc. CASoN 2013*, pp. 21–26.
- [26] S. Tirthapura & D.P. Woodruff. A general method for estimating correlated aggregates over a data stream. In *Proc. IEEE ICDE 2012*, pp. 162–173.
- [27] E. Valari, M. Kontaki, & A.N. Papadopoulos. Discovery of top-k dense subgraphs in dynamic graph collections. In *Proc. SSDBM 2012*, pp. 213–230.
- [28] F. Wei-Kleiner. Finding nearest neighbors in road networks: a tree decomposition method. In *Proc. EDBT/ICDT 2013 Workshops (GraphQ)*, pp. 233–240.

# Linked Web Data Management (LWDM)

Devis Bianchini (Università degli Studi di Brescia, Italy)  
Valeria De Antonellis (Università degli Studi di Brescia, Italy)  
Roberto De Virgilio (University of Rome, Italy)



# Quantifying the Connectivity of a Semantic Warehouse

Yannis Tzitzikas<sup>1,2</sup>, Nikos Minadakis<sup>1</sup>, Yannis Marketakis<sup>1</sup>,  
Pavlos Fafalios<sup>1,2</sup>, Carlo Allocca<sup>1</sup>, Michalis Mountantonakis<sup>1,2</sup>

<sup>1</sup> Institute of Computer Science, FORTH-ICS, GREECE, and

<sup>2</sup> Computer Science Department, University of Crete, GREECE

{tzitzik,minadakn,marketak,fafalios,carlo,mountant}@ics.forth.gr

## ABSTRACT

In many applications one has to fetch and assemble pieces of information coming from more than one SPARQL endpoints. In this paper we describe the corresponding requirements and challenges, and then we present a process for constructing such a semantic warehouse. We focus on the aspects of *quality* and *value* of the warehouse, and for this reason we introduce various metrics for quantifying its *connectivity*, and consequently its ability to answer complex queries. We demonstrate the behavior of these metrics in the context of a real and operational semantic warehouse. The results are very promising: the proposed metrics-based matrixes allow someone to get an overview of the contribution (to the warehouse) of each source and to quantify the benefit of the entire warehouse. The later is useful also for monitoring the quality of the warehouse after each reconstruction.

## 1. INTRODUCTION

An increasing number of datasets are already available as Linked Data. For exploiting this wealth of data, and building domain specific applications, in many cases there is the need for fetching and assembling pieces of information coming from more than one SPARQL endpoints. These pieces are then used for constructing a *warehouse*, for offering more complete browsing and query services (in comparison to those offered by the underlying sources).

We shall use the term *Semantic Warehouse* (for short warehouse) to refer to a read-only set of RDF triples fetched (and transformed) from different sources that aims at serving a particular set of query requirements.

We can distinguish *domain independent* warehouses, like the Sindice RDF search engine [9], or the Semantic Web Search Engine (SWSE) [4], but also *domain specific*, like TaxonConcept<sup>1</sup> and marineTLO-based warehouse [12].

In this paper we focus on the requirements for building domain specific semantic warehouses. Such warehouses aim to

<sup>1</sup><http://www.taxonconcept.org/>

serve particular needs, for particular communities of users, consequently their “quality” requirements are more strict. It is therefore worth elaborating on the process that can be used for building such warehouses, and on the related difficulties and challenges. In brief, for building such a warehouse one has to tackle various challenges and questions, e.g. how to define the objectives and the scope of such a warehouse, how to *connect* the fetched pieces of information (common URIs or literals are not always there), how to tackle the various issues of provenance that arise, how to keep the warehouse fresh, i.e. how to automate its construction or refreshing. In this paper we focus on the following questions:

- How to measure the value and quality (since this is important for e-science) of the warehouse?
- How to monitor its quality after each reconstruction or refreshing (as the underlying sources change)?

We have encountered these questions in the context of a real semantic warehouse for the *marine* domain which harmonizes and connects information from different sources of marine information. Past works have focused on the notion of conflicts, and have not paid attention to connectivity. We use the term *connectivity* to express the degree up to which the contents of the semantic warehouse form a connected graph that can serve, ideally in a correct and complete way, the query requirements of the semantic warehouse, while making evident how each source contributes to that degree. To this end in this paper we introduce and evaluate several metrics for quantifying the connectivity of the warehouse. These metrics allow someone to get an overview of the contribution (to the warehouse) of each source (enabling the discrimination of the important from the non important sources) and to quantify the benefit of such a warehouse

The paper is organized as follows: Section 2 describes the main requirements, related works, and what distinguishes the current work. Section 3 provides the context by describing the process used for constructing such warehouses. Section 4 introduces the quality metrics and demonstrates their use. Finally, Section 5 concludes the paper.

## 2. REQUIREMENTS AND RELATED WORK

The context of this work is the ongoing *iMarine* project<sup>2</sup> that offers an operational distributed infrastructure that serves hundreds of scientists from the marine domain. As regards semantically structured information, the objective is to integrate information from various marine sources, specif-

<sup>2</sup>FP7, Research Infrastructures, <http://www.i-marine.eu/>

ically from WoRMS<sup>3</sup>, Ecoscope<sup>4</sup>, FishBase<sup>5</sup>, FLOD<sup>6</sup> and DBpedia<sup>7</sup>.

The integrated warehouse (its first version is described in [12])<sup>8</sup> is now operational and it is exploited in various applications, including generators of fact sheets (e.g. TunaAtlas<sup>9</sup>), or for enabling exploratory search services (e.g. X-ENS [3] that offers semantic post-processing of search results). Below we list the main functional and non functional requirements for constructing such warehouses.

### Functional Requirements

- *Multiplicity of Sources.* Ability to query SPARQL endpoints (and other sources), get the results, and ingest them to the warehouse.
- *Mappings, Transformations and Equivalences.* Ability to accommodate schema mappings, perform transformations and create **sameAs** relationships between the fetched content for connecting the corresponding schema elements and entities.
- *Reconstructibility.* Ability to reconstruct the warehouse periodically (from scratch or incrementally) for keeping it fresh.

### Non Functional Requirements

- *Scope control.* Make concrete and testable the scope of the information that should be stored in the warehouse. Since we live in the same universe, everything is directly or indirectly connected, therefore without stating concrete objectives there is the risk of continuous expansion without concrete objectives regarding its contents, quality and purpose.
- *Connectivity assessment.* Ability to check and assess the connectivity of the information in the warehouse. Putting triples together does not guarantee that they will be connected. In general, connectivity concerns both schema and instances and it is achieved through common URIs, common literals and **sameAs** relationships. Poor connectivity affects negatively the query capabilities of the warehouse. Moreover, the contribution of each source to the warehouse should be measurable, for deciding which sources to keep or exclude (there are already hundreds of SPARQL endpoints).
- *Provenance.* More than one levels of provenance can be identified and are usually required, e.g. warehouse provenance (from what source that triple was fetched), information provenance (how the fact that the  $x$  species is found in  $y$  water area was produced), and query provenance (which sources and how contributed to the answer of this query).
- *Consistency and Conflicts.* Ability to specify the desired consistency level of the warehouse e.g. do we want to tolerate an association between a fish commercial code and more than one scientific names? Do we want to consider this as inconsistency (that makes

the entire warehouse, or parts of it, unusable), or as resolvable (through a rule) conflict, or as a normal case (and allow it as long as the provenance is available).

## 2.1 Related Approaches

Below we refer and discuss in brief the more related systems, namely *ODCleanStore* and *Sieve*.

*ODCleanStore* [8, 6, 5] is a tool that can download content (RDF graphs) and offers various transformations for cleaning it (deduplication, conflict resolution), and linking it to existing resources, plus assessing the quality of the outcome. It names *conflicts* the cases where two different quads (e.g. sources) have different object values for a certain subject  $s$  and predicate  $p$ . To such cases conflict resolution rules are offered that either select one or more of these conflicting values (e.g. ANY, MAX, ALL), or compute a new value (e.g. AVG). [5] describes various quality metrics (for scoring each source based on conflicts), as well for assessing the overall outcome.

Another related system is *Sieve* [7] which is part of the Linked Data Integration Framework (LDIF)<sup>10</sup>. This work proposes metrics like *schema completeness* and *conciseness*. However, such metrics are not useful for the case of domain specific warehouses that have a top-level ontology, in the sense that the schema mappings and the transformation rules can tackle these problems. This is true in our warehouse (it is also assumed in the scenarios of *ODCleanStore*).

Overall, we can say that the quality metrics introduced by other works focus more on conflicts. The aspect of connectivity, is not covered sufficiently. The aspect of connectivity is important in warehouses whose schema is not small, and consequently the queries contain paths. The longer such paths are, the more the query capability of the warehouse is determined by the connectivity.

Of course, the issue of data warehouse quality is older than the RDF world, e.g. [10, 1]. A discussion of related works for the RDF world is available in [2], that also focuses on describing data sources in terms of their completeness in query answering. Another quality perspective identified in [13] is that of the *specificity* of the ontology-based descriptions under ontology evolution, an issue that is raised when ontologies and vocabularies evolve over time.

Finally, we could mention that works like [11], which focus on the statistical evaluation of the metadata elements of a repository, are not directly related, since they do not consider the characteristics of RDF and Linked Data, nor they try to evaluate the contribution of the underlying sources.

## 3. THE INTEGRATION PROCESS

For making clear the context, here we describe in brief the steps of the process that we follow for creating the warehouse. Figure 1 shows an overview of the warehouse's contents, while Figure 2 sketches the construction process<sup>11</sup>.

The first step is to *define requirements* in terms of *competency queries*. It is a set of queries (provided by the community) indicating the queries that the warehouse is intended to serve. Some indicative queries are given in Appendix A,

<sup>3</sup><http://www.marinespecies.org/>

<sup>4</sup><http://www.ecoscopebc.ird.fr/EcoscopeKB/ShowWelcomePage.action>

<sup>5</sup><http://www.fishbase.org/>

<sup>6</sup><http://www.fao.org/figis/flod/>

<sup>7</sup><http://dbpedia.org/>

<sup>8</sup>URL of the warehouse (restricted access): <http://virtuoso.i-marine.d4science.org:8890/sparql>

<sup>9</sup><http://vmecoscopebc-protompl.ird.fr:8080/semantic-atlas/ShowWelcomePage>

<sup>10</sup><http://www4.wiwiw.fu-berlin.de/bizer/ldif/>

<sup>11</sup>Extra material is available at <http://www.ics.forth.gr/isl/MarineTLO/#applications>.

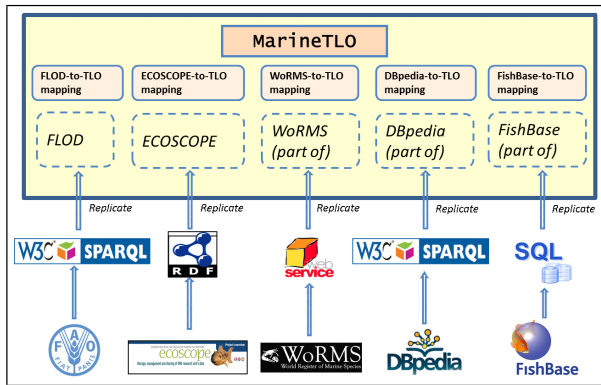


Figure 1: Overview of the warehouse

the full list is web accessible<sup>12</sup>. It is always a good practice to have (select or design) a *top-level schema/ontology* as it alleviates the schema mapping effort (avoids the combinatorial explosion of pair-wise mappings) and allows formulating the competency queries using that ontology (instead of using elements coming from the underlying sources, which change over time). For our case in iMarine, the ontology is called *MarineTLO* [12]<sup>13</sup>.

The next step is to *fetch the data* from each source and this requires using various access methods (SPARQL endpoints, HTTP accessible files, JDBC) and specifying what exactly to get from each source (all contents or a specific part). For instance, and for the case of the iMarine warehouse, we fetch all triples from FLOD through its SPARQL endpoint, all triples from Ecoscope obtained by fetching OWL files from its web page, information about species (ranks, scientific and common names) from WoRMS, information about species from DBpedia’s SPARQL endpoint, and finally information about species, water areas, ecosystems and countries from the relational tables of FishBase.

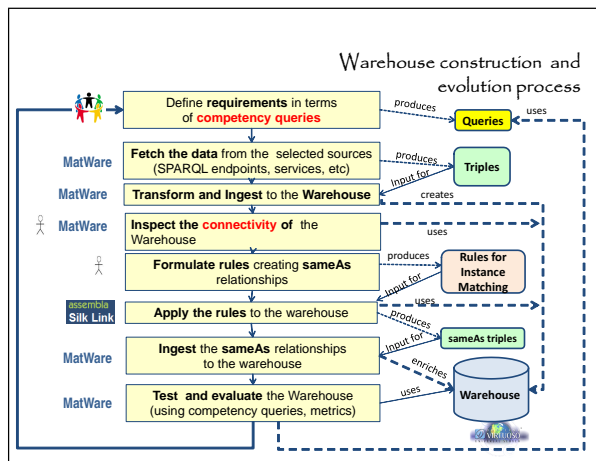


Figure 2: The process for constructing and evolving the warehouse

The next step is to *transform and ingest* the fetched data.

<sup>12</sup>[http://www.ics.forth.gr/isl/MarineTLO/competency\\_queries/MarineTLO\\_Competency\\_Queries\\_Version\\_v3.pdf](http://www.ics.forth.gr/isl/MarineTLO/competency_queries/MarineTLO_Competency_Queries_Version_v3.pdf)

<sup>13</sup><http://www.ics.forth.gr/isl/MarineTLO>

Some data can be stored as they are fetched, while others have to be transformed, i.e. a *format* transformation and/or a *logical* transformation has to be applied for being compatible with the top-level ontology. For example, a format transformation may be required to transform information expressed in DwC-A (a format for sharing biodiversity data), to RDF. A logical transformation may be required for transforming a string literal to a URI, or for splitting a literal for using its constituents, or for creating intermediate nodes (e.g. instead of  $(x, \text{hasName}, y)$  to have  $(x, \text{hasNameAssignment}, z), (z, \text{name}, y), (z, \text{date}, d)$ , etc.

This step also includes the definition of the required *schema mappings* that are required for associating the fetched data with the schema of the top level ontology. Another important aspect for domain specific warehouses, is the management of provenance. In our case we support what we call “warehouse”-provenance, i.e. we store the fetched (or fetched and transformed) triples from each source in a separate *graphspace* (a graphspace is a named set of triples which can be used for restricting queries and updates in a RDF triple store). In this way we know which source has provided what facts and this is exploitable also in the queries. As regards conflicts (e.g. different values for the same properties), the adopted policy in our case is to make evident the different values and their provenance, instead of making decisions, enabling in this way the users to select the desired values, and the content providers to spot their differences. The adoption of separate graphspaces also allows refreshing parts of the warehouse, i.e. the part that corresponds to one source. Furthermore, it makes feasible the computation of the metrics that are introduced in the next section.

The next step is to *inspect and test the connectivity* of the “draft” warehouse. This is done through the competency queries as well as through the metrics that we will introduce. The former (competency queries) require manual inspection, but automated tests are also supported. In brief, let  $q$  be a query in the set of competency queries. Although we may not know the “ideal” answer of  $q$ , we may know that it should certainly contain a particular set of resources, say  $Pos$ , and should not contain a particular set of resources, say  $Neg$ . Such information allows automated testing. If  $ans(q)$  is the answer of  $q$  as produced by the warehouse, we would like to hold  $Pos \subseteq ans(q)$  and  $Neg \cap ans(q) = \emptyset$ . Since these conditions may not hold, it is beneficial to adopt an IR-inspired evaluation, i.e. compute the *precision* and *recall* defined as:  $precision = 1 - \frac{|Neg \cap ans(q)|}{|ans(q)|}$ ,  $recall = \frac{|Pos \cap ans(q)|}{|Pos|}$ . The bigger the values we get the better (ideally 1). The better we know the desired query behaviour, the bigger the sets  $Pos$  and  $Neg$  are, and consequently the more safe the results of such evaluation are.

Based also on the results of the previous step, the next step is to *formulate rules for instance matching*, i.e. rules that can produce *sameAs* relationships for obtaining the desired connections. For this task we employ the tool SILK[14]<sup>14</sup>. Then, we *apply the instance matching rules* (SILK rules in our case) for producing (and then ingesting to the warehouse) *sameAs* relationships.

Finally we have to test the produced repository and evaluate it. This is done through the competency queries and through the metrics that we will introduce.

**Periodic Reconstruction** Above we have described the

<sup>14</sup><http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/>

steps required for the first time. After that the warehouse is reconstructed periodically for getting refreshed content. This is done *automatically* through a tool that we have developed called **MatWare**. The metrics that we will introduce are very important for *monitoring* the warehouse after reconstructing it. For example by comparing the metrics in the past and new warehouse, one can understand whether a change in the underlying sources affected negatively the quality (e.g. connectivity) of the warehouse.

## 4. CONNECTIVITY METRICS

The objective is to define *metrics* for assisting humans on assessing in concrete terms the quality and the value offered by the warehouse.

To aid understanding, after defining each metric we show the values of these metrics as computed over the iMarine warehouse which is built using data from FLOD, WoRMS, Ecoscope, DBpedia, and FishBase. The warehouse is real and operational<sup>15</sup>. This also allows testing whether the metrics are successful.

At first we introduce some required notations. Let  $S = S_1, \dots, S_k$  be the set of underlying sources. Each contributes to the warehouse a set of triples (i.e. a set of subject-predicate-object statements), denoted by  $triples(S_i)$ . This is not the set of all triples of the source. It is the subset that is contributed to the warehouse (fetched mainly by running SPARQL queries). We shall use  $U_i$  to denote the URIs that appear in the triples in  $triples(S_i)$ . Hereafter, we consider only those URIs that appear as *subjects* or *objects* in a triple. We do not include the URIs of the properties because they concern the schema and this integration aspect is already tackled by the top level schema.

Let  $W$  denote the triples in the warehouse, i.e.  $W = \cup_{1..k} triples(S_i)$ .

### On Comparing URIs

For computing the metrics that are defined next, we need methods to compare URIs coming from different sources. There are more than one methods, or policies, for doing so. Below we distinguish three main policies:

- i *Exact String Equality*. We treat two URIs  $u_1$  and  $u_2$  as equal, denoted by  $u_1 \equiv u_2$ , if  $u_1 = u_2$  (i.e. strings equality).
- ii *Suffix Canonicalization*. Here we consider that  $u_1 \equiv u_2$  if  $last(u_1) = last(u_2)$  where  $last(u)$  is the string obtained by (a) getting the substring after the last "/" or "#", and (b) turning the letters of the picked substring to lowercase and deleting the underscore letters that might exist. According to this policy  
`http://www.dbpedia.com/Thunnus_Albacares`  $\equiv$   
`http://www.ecoscope.com/thunnus_albacares`  
 since their canonical suffix is the same, i.e. `thunnusalbacares`. Another example of a equivalent URIs:  
`http://www.s1.com/entity#thunnus_albacares`  $\equiv$   
`http://www.s2.org/entity/thunnusAlbacares`.
- iii *Entity Matching*. Here consider  $u_1 \equiv u_2$  if  $u_1$  **sameAs**  $u_2$  according to the entity matching rules that are (or will be eventually) used for the warehouse. In general

such rules create **sameAs** relationships between URIs. In our case we use SILK for formulating and applying such rules.

Note that if two URIs are equivalent according to policy [i], then they are equivalent according to [ii] too. Policy [i] is very strict (probably too strict for matching entities coming from different sources), however it does not produce any false-positive. Policy [ii] achieves treating as equal entities across different namespaces, however false-positives may occur. For instance, **Argentina** is a *country* (`http://www.fishbase.org/entity#Argentina`) but also a *fish genus* (`http://www.marinespecies.org/entity#WoRMS:125885/Argentina`). Policy [iii] is fully aligned with the intended query behaviour of the warehouse (the formulated rules are expected to be better as regards false-negatives and false-positives), however for formulating and applying these entity matching rules, one has to know the contents of the sources. Consequently one cannot apply policy [iii] the first time, instead policies [i] and [ii] can be applied automatically without requiring any human effort. We could also note that policy [ii] can be used for providing hints regarding what entity matching rules to formulate.

Below we define and compute the metrics assuming policy [ii], i.e. whenever we have a set operation we assume equivalence according to [ii] (e.g.  $A \cap B$  means  $\{a \in A \mid \exists b \in B \text{ s.t. } a \equiv_{[ii]} b\}$ ). Then, in Section 4.1, we report results according to policy [iii].

### Matrix of Percentages of Common URIs

The number of *common URIs* between two sources  $S_i$  and  $S_j$ , is given by  $|U_i \cap U_j|$ . We can define the *percentage of common URIs* (a value ranging [0..1]), as follows:  $curi_{i,j} = \frac{|U_i \cap U_j|}{\min(|U_i|, |U_j|)}$ . In the denominator we use  $\min(|U_i|, |U_j|)$  instead of  $|U_i \cup U_j|$  as in the Jaccard similarity. With Jaccard similarity the integration of a small triple set with a big one would always give small values, even if the small set contains many URIs that exist in the big set. For this reason we use *min*.

We now extend the above metric and consider *all* sources aiming at giving an overview of the warehouse. Specifically, we compute a  $k \times k$  matrix where  $c_{i,j} = curi_{i,j}$ . The higher values this matrix contains, the more glued its "components" are. However note that we may have 3 sources, such that each pair of them has a high *curi* value, but the intersection of the URIs of all 3 sources is empty. This is not necessarily bad, for example, consider a source contributing triples of the form `person-lives-placeName`, a second source contributing `placeName-has-postalCode`, and a third one contributing `postCode-isAddressOf-cinema`. Although these three sources may not contain even one common URI, their hosting in a warehouse allows answering queries: "*give me the cinemas in the area where the x person lives*".

On the other hand, in a case where the three sources were contributing triples of the form `person-lives-placeName`, `person-worksAt-Organization` and `person-owns-car`, then it would be desired to have common URIs in all sources, as that would allow having more complete information for many persons. Finally, one might wonder why we do not introduce a kind of average path length, or diameter, for the warehouse. Instead of doing that, we inspect the paths that are useful for answering the queries of the users, and this is done through the competency queries.

For the warehouse at hand, Table 1 shows the matrix of

<sup>15</sup>In the evaluation of related tools, like *Sieve* [7] and *OD-CleanStore* [8], real datasets have been used but not "real" operational needs. In our evaluation we use an operational warehouse with concrete (query) requirements which are described by the competency queries.

the common URIs, while Table 2 shows the matrix of the common URI percentages. The percentages range from 0.3% to 27.39%. We can see that in some cases we have a significant percentage of common URIs between the different sources. The biggest intersection is between FishBase and DBpedia.

$S_i \backslash S_j$	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	173,929	239	523	631	887
WoRMS		80,485	200	1,714	3,596
Ecoscope			5,824	192	225
DBpedia				70,246	9,578
FishBase					34,974

Table 1: Common URIs ( $|U_i \cap U_j|$ )

$S_i \backslash S_j$	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	1	0.3%	8.98%	0.9%	2.54%
WoRMS		1	3.43%	2.44%	10.28%
Ecoscope			1	3.3%	3.86%
DBpedia				1	27.39%
FishBase					1

Table 2: Common URIs % ( $curi_{i,j} = \frac{|U_i \cap U_j|}{\min(|U_i|, |U_j|)}$ )

### Percentage of Common literals between two sources

The *percentage of common literals*, between two sources  $S_i$  and  $S_j$  can be computed by  $clit_{i,j} = \frac{|Lit_i \cap Lit_j|}{\min(|Lit_i|, |Lit_j|)}$ . To compare 2 literals coming from different sources, we convert them to lower case, to avoid cases like comparing “Thunnus” from one source and “thunnus” from another.

Table 3 shows the matrix of the common literals, while Table 4 shows the percentages. We can see that as regards the literals the percentages of similarity are even smaller than the ones regarding common URIs. The percentages range from 2.71% to 12.37%.

$S_i \backslash S_j$	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	111,164	3,624	1,745	5,668	9,505
WoRMS		51,076	382	2,429	4,773
Ecoscope			14,102	389	422
DBpedia				123,887	14,038
FishBase					138,275

Table 3: Common Literals ( $|Lit_i \cap Lit_j|$ )

### Increase in the Average Degree

Now we introduce another metric for expressing the degree of common URIs. Let  $E$  be the entities of interest (or all URIs). If  $T$  is a set of triples, then we can define the *degree* of an entity  $e$  in  $T$  as:  $deg_T(e) = |\{(s, p, o) \in T \mid s = e \text{ or } o = e\}|$ , while for a set of entities  $E$  we can define their average degree in  $T$  as  $deg_T(E) = avg_{e \in E}(deg_T(e))$ .

Now for each source  $S_i$  we can compute the average degree of the elements in  $E$  considering  $triples(S_i)$ . If the sources of the warehouse contain common elements of  $E$ , then if we compute the degrees in the graph of  $W$  (i.e.  $deg_W(e)$  and  $deg_W(E)$ ), we will get higher values. So the increase in the degree is a way to quantify the gain, in terms of connectivity, that the warehouse offers.

For each source  $S_i$ , Table 5 shows the average degree of its URIs (i.e. of those in  $U_i$ ), and the average degree of the same

$S_i \backslash S_j$	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	1	7.1%	12.37%	5.1%	8.55%
WoRMS		1	2.71%	4.76%	9.34%
Ecoscope			1	2.76%	2.99%
DBpedia				1	11.33%
FishBase					1

Table 4: Common Literals % ( $clit_{i,j} = \frac{|Lit_i \cap Lit_j|}{\min(|Lit_i|, |Lit_j|)}$ )

$S_i$	avg $deg_{S_i}(U_i)$	avg $deg_W(U_i)$	increase
FLOD	7.18	9.18	27.84%
WoRMS	3.3	7.33	122.36%
Ecoscope	22.84	31.18	36.56%
DBpedia	41.41	42.11	1.7%
FishBase	18.86	29.81	58.08%
<b>AVERAGE</b>	18.72	23.92	27.78%

Table 5: Average degrees in sources and in the warehouse

URIs in the warehouse graph. It also reports the increment percentage (computed by warehouse/source \* 100). The last row of the table shows the average values of each column. We observe that the average degree is increased from 18.72 to 23.92

### Restricting the Metrics (to the Entities of Interest)

The above metrics can be refined so that to consider not all URIs, but only those that serve the purpose of the warehouse. For example, one could define the above metrics by considering only URIs that are instances of a particular class or classes (e.g. Persons, Locations), or those returned by the competency queries. In general, we can consider that the set of URIs (or entities) of interest is a set  $E$  that is defined extensionally (by listing its elements) or intentionally (through a query).

### Complementarity of Sources

We now define metrics for quantifying the complementarity of the sources.

The “contribution” of each source  $S_i$  can be quantified by counting the triples it has provided to the warehouse, i.e. by  $|triples(S_i)|$ . We can also define its “unique contribution” by excluding from  $triples(S_i)$  those belonging to the triples returned by the other sources. Formally, we can define  $triplesUnique(S_i) = triples(S_i) \setminus (\cup_{1 \leq j \leq k, j \neq i} triples(S_j))$ . It follows that if a source  $S_i$  provides triples which are also provided by other sources, then we have  $triplesUnique(S_i) = \emptyset$ . Consequently, and for quantifying the contribution of each source to the warehouse, we can compute and report the number of its triples  $|triples(S_i)|$ , the number of unique triples  $|triplesUnique(S_i)|$ , and the percentage of unique triples  $\frac{|triplesUnique(S_i)|}{|triples(S_i)|}$ . To count the unique triples of each source, for each triple of that source we perform suffix canonicalization on its URIs, convert its literals to lower case, and then we check if the resulting (canonical) triple exists in the canonical triples of a different source. If not, we count this triple as unique.

Let  $triplesUniques$  be the union of the unique triples of all sources, i.e.  $triplesUniques = \cup_i triplesUnique(S_i)$ . This set can be proper subset of  $W$  (i.e.  $triplesUniques \subset W$ ), since it does not contain triples which have been contributed by two or more sources.

Table 6 shows for each source the number of its triples

$|triples(S_i)|$ , the number of unique triples  $|triplesUnique(S_i)|$ , and the percentage of unique triples  $\frac{|triplesUnique(S_i)|}{|triples(S_i)|}$ . We can see that every source contains a very high (> 99%) percentage of unique triples, so we can conclude that all sources are important.

$S_i$	$a =  triples(S_i) $	$b =  triplesUnique(S_i) $	$b/a$
FLOD	665,456	664,703	99.89%
WoRMS	461,230	460,741	99.89%
Ecoscope	54,027	53,641	99.29%
DBpedia	450,429	449,851	99.87%
FishBase	1,425,283	1,424,713	99.96%

**Table 6: (Unique) triple contributions of the sources**

We now define another metric for quantifying the value of the warehouse for the entities of interest. Specifically we define the *complementarity factor for an entity  $e$* , denoted by  $cf(e)$ , as the number of sources that provided unique material about  $e$ . It can be defined declaratively as:

$$cf(e) = |\{ i \mid triples_w(e) \cap triplesUnique(S_i) \neq \emptyset \}|$$

i.e. it is the number of sources which have provided unique content for  $e$ . Note that if  $k = 1$ , i.e. if we have only one source, then for every entity  $e$  we will have  $cf(e) = 1$ . If  $k = 2$ , i.e. if we have two sources, then we can have the following cases:

- $cf(e) = 0$  if both sources have provided the same triple (or triples) about  $e$ ,
- $cf(e) = 1$  if the triples provided by the one source (for  $e$ ) are subset of the triples provided by the other,
- $cf(e) = 2$  if each source has provided at least one different triple for  $e$  (of course they can also have contributed common triples).

Consequently for the entities of interest we can compute and report the average *complementarity factor* as a way to quantify the value of the warehouse for these entities.

Table 7 shows (indicatively) the *complementarity factors* for a few entities which are important for the problem at hand. We see that for the entities “Thunnus” and “Shark” each source provides unique information (with the term entity we mean any literal or URI that contains the word “thunnus” for example). For the entity “Greece” and “Astrapogon” we take unique information from three sources. The fact that the complementarity factor is big means that the warehouse provides information about each entity from all/many sources.

Kind of Entity	$cf(\cdot)/5$
Thunnus	5/5
Greece	3/5
Shark	5/5
Astrapogon	3/5

**Table 7: Complementarity factor ( $cf$ ) of some entities**

#### 4.1 After applying the rule-derived ‘sameAs’ relationships and the transformation rules

So far in the computation of the above metrics we have used policy [ii] (suffix canonicalized URIs) when comparing URIs. Here we show the results from computing again these metrics using policy [iii]. This means that now when comparing URIs we consider the **sameAs** relationships that have

been produced by the entity matching rules of the warehouse. In the current warehouse we use 11 SILK rules. An indicative SILK rule is the following: “If an Ecoscope individual’s attribute prelabel (e.g. *Thunnus albacares*) in lower case is the same with the attribute label in latin of a FLOD individual (e.g. ‘thunnus albacares’@la), then these two individuals are the same”.

We should also note that previously, in policy [ii] we considered the triples as they are fetched from the sources. Here we consider the triples as derived from the transformation rules (described in §3).

Computing the metrics using policy [iii], not only allows evaluating the gain achieved by these relationships, but it also better reflects the value of the warehouse since query answering considers the **sameAs** relationships.

Table 8 shows the matrix of the common URIs after the rule-derived **sameAs** relationships and the execution of the transformation rules, and Table 9 shows the corresponding percentages. We can see that, compared to the results of Tables 1 and 2, after considering the **sameAs** relationships the number of common URIs between the different sources is significantly increased (more than 7 times in some cases).

$S_i \backslash S_j$	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	190,733	434	1,897	4,009	6,732
WoRMS		80,486	805	1,754	3,596
Ecoscope			7,805	1,245	2,116
DBpedia				74,381	10,385
FishBase					34,974

**Table 8: Common URIs ( $|U_i \cap U_j|$ )**

$S_i \backslash S_j$	FLOD	WoRMS	Ecoscope	DBpedia	FishBase
FLOD	1	0.54%	24.3%	5.39%	19.25%
WoRMS		1	10.31%	2.36%	10.28%
Ecoscope			1	15.95%	27.1%
DBpedia				1	29.69%
FishBase					1

**Table 9: Common URIs % ( $curi_{i,j} = \frac{|U_i \cap U_j|}{\min(|U_i|, |U_j|)}$ )**

Table 10 shows the average degree of the URIs of each source  $S_i$  (i.e. of those in  $U_i$ ), and the average degree of the same URIs in the warehouse graph. It also reports the increment percentage (computed by warehouse/source \* 100). The last row of the table shows the average values of each column. We can see that the average degree, of all sources, after the inclusion of the **sameAs** relationships is significantly bigger than before. In comparison to Table 5, the increase is from 2 to almost 8 times bigger. This means that we achieve a great increase in terms of the connectivity of the information in the warehouse.

As regards the unique contribution of each source, Table 11 shows the number of the triples of each source  $|triples(S_i)|$ , the number of unique triples  $|triplesUnique(S_i)|$ , and the percentage of unique triples  $\frac{|triplesUnique(S_i)|}{|triples(S_i)|}$ . We observe that the values in the column “ $a$ ” are increased in comparison to Table 6. This is because of the execution of the transformation rules after the ingestion of the data to the warehouse, which results to the creation of new triples for the majority of sources. Finally we observe that, in general,

$S_i$	avg $deg_{S_i}(U_i)$	avg $deg_w(U_i)$	increase
FLOD	7.18	54.31	656.51%
WoRMS	3.3	9.93	201.36%
Ecoscope	22.84	165.24	623.6%
DBpedia	41.41	84.2	103.36%
FishBase	18.86	50.6	168.32%
<b>AVERAGE</b>	<b>18.72</b>	<b>72.86</b>	<b>289.21%</b>

Table 10: Average degrees in sources and in the warehouse

the percentage of unique triples provided by each source is decreased. This happens because the transformation rules and the same-as relationships have turned previously different triples, the same.

$S_i$	$a =  \text{triples}(S_i) $	$b =  \text{triplesUnique}(S_i) $	$b/a$
FLOD	810,301	798,048	98.49%
WoRMS	582,009	527,358	99.88%
Ecoscope	138,324	52,936	38.27%
DBpedia	526,016	517,242	98.33%
FishBase	1,425,283	1,340,968	94.08%

Table 11: (Unique) triple contributions of the sources

## 4.2 Detecting Redundancies or other Pathological Cases

The metrics can be used also for detecting various pathological cases, e.g. sources that do not have any common URI or literal, or “redundant sources”. To test this we created two artificial sources, let’s call them *Airports* and *CloneSource*. The first contains triples about airports which were fetched from the DBpedia public SPARQL endpoint, while the second is a subset of Ecoscope’s and DBpedia’s triples as they are stored in the warehouse.

In the sequel, we computed the metrics for all 7 sources. Table 12 shows the unique triples and Table 13 shows the average degrees. As regards *Airports*, the percentage of common URIs was very low, and the average degree for the entities of that source was very low too (2.22% due to some common country names), while its unique contribution was 100%. As regards *CloneSource* we got 0 unique contribution (as expected, since it was composed from triples of existing sources).

$S_i$	$a =  \text{triples}(S_i) $	$b =  \text{triplesUnique}(S_i) $	$b/a$
FLOD	665,456	664,703	99.89%
WoRMS	461,230	460,741	99.89%
Ecoscope	54,027	17,951	33.23%
DBpedia	450,429	429,426	95.34%
Fishbase	1,425,283	1,424,713	99.96%
<i>CloneSource</i>	56,195	0	0%
<i>Airports</i>	31,628	31,628	100%

Table 12: (Unique) triple contributions of the sources

### Rules for Detecting Pathological Cases

It follows that we can detect pathological cases using two rules: (a) if the average increase of the degree of the entities of a source is low, then this means that its contents are not connected with the contents of the rest sources (this is the

case of *Airports* where we had only 2.22% increase), (b) if the unique contribution of a source is very low (resp. zero), then this means that it does not contribute significantly (resp. at all) to the warehouse (this is the case of *CloneSource* where the unique contribution was zero).

$S_i$	avg $deg_{S_i}(U_i)$	avg $deg_w(U_i)$	increase
FLOD	7.18	54.31	656.51%
WoRMS	3.3	9.93	201.36%
Ecoscope	22.84	165.24	623.6%
DBpedia	41.41	84.2	103.36%
FishBase	18.86	50.6	168.32%
<i>CloneSource</i>	44.43	84.2	89.52%
<i>Airports</i>	70.99	72.56	2.22%
<b>AVERAGE</b>	<b>29.86</b>	<b>74.43</b>	<b>149.26%</b>

Table 13: Average degrees in sources and in the warehouse

## 4.3 Implementation

As regards implementation, the above metrics are computed by the tool *MatWare* that we have developed. The values of the metrics are exposed in the form of an HTML page (as shown in Figure 3) providing in this way a kind of quantitative documentation of the warehouse. As regards time, the current warehouse (containing 3,772,919 triples) takes about 7 hours to reconstruct.<sup>16</sup>

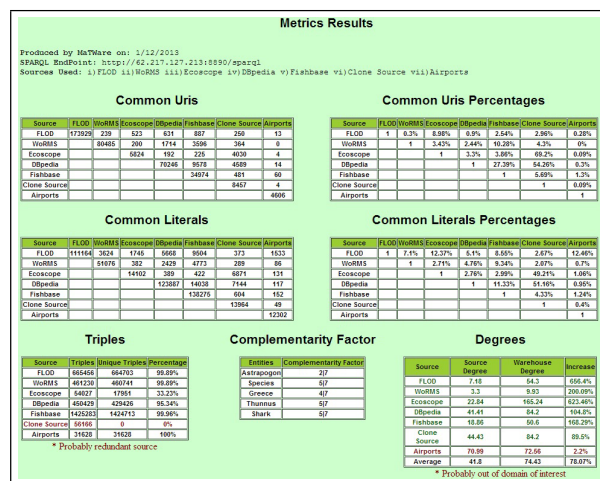


Figure 3: Metrics results displayed in HTML as produced by MatWare

## 5. SYNOPSIS AND CONCLUSION

For many applications one has to fetch and assemble pieces of information coming from more than one SPARQL endpoints. In this paper we have described the main requirements and challenges, based also on our experience so far in building a semantic warehouse for marine resources. We have presented a process for constructing such warehouses and then we introduced metrics for quantifying the connectivity of the outcome.

The results are very positive. By inspecting the proposed metrics-based matrixes one can very quickly get an overview

<sup>16</sup>Virtuoso and machine spec: Openlink Virtuoso V6.1, Ubuntu 12.10 64bit, Quad-Core, 4 GB RAM.



of the contribution of each source and the tangible benefits of the warehouse. The main metrics proposed are: (a) the matrix of percentages of the common URIs and/or literals, (b) the complementarity factor of the entities of interest, (c) the table with the increments in the average degree of each source, and (d) the unique triple contribution of each source. The values of (a),(b),(c) allow valuating the warehouse, while (c) and (d) mainly concern each particular source.

For instance, and for the warehouse at hand, by combining the unique triples contribution (from Table 11) and the increment of the average degrees (of Table 10), we can understand that not only we get unique information *from all* sources, but also *how much* the average degree of the entities of the sources has been increased in the warehouse. Moreover, redundant sources can be spotted through their low unique contribution, while unconnected sources through their low average increase of the degree of their entities. Of course one could combine the above metrics and derive various other single-valued metrics for expressing the quality (connectivity, redundancy) of each source, as well as for the entire warehouse.

The ability to assess the quality of a semantic warehouse (using methods like those presented in this paper, as well those presented in §2.1) is very important for judging whether the warehouse can be used in e-Science. It is also important because in the long run we expect that datasets and warehouses will be peer-reviewed, evaluated and cited, and this in turn will justify actions for their future preservation.

In future we plan to continue along this direction, focusing also on methods that compare the metrics of two different warehouse versions for monitoring the evolution of the warehouse over time.

## Acknowledgement

This work was partially supported by the ongoing project *iMarine* (FP7 Research Infrastructures, 2011-2014).

## 6. REFERENCES

- [1] D. P. Ballou and G. K. Tayi. Enhancing data quality in data warehouse environments. *Communications of the ACM*, 42(1):73–78, 1999.
- [2] F. Darari, W. Fariz, W. Nutt, G. Pirro, and S. Razniewski. Completeness Statements about RDF Data Sources and their Use for Query Answering. In *The Semantic Web–ISWC 2013*, pages 66–83. Springer, 2013.
- [3] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In *SIGIR’13*, pages 1089–1090, Dublin, Ireland, 2013. ACM.
- [4] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and Browsing Linked Data with SWSE: The Semantic Web Search Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4), 2011.
- [5] T. Knap and J. Michelfeit. Linked Data Aggregation Algorithm: Increasing Completeness and Consistency of Data, <http://www.ksi.mff.cuni.cz/~knap/files/aggregation.pdf>.
- [6] T. Knap, J. Michelfeit, J. Daniel, P. Jerman, D. Rychnovský, T. Soukup, and M. Nečaský. ODCleanStore: a Framework for Managing and Providing Integrated Linked Data on the Web. In *Web Information Systems Engineering-WISE 2012*, pages 815–816. Springer, 2012.
- [7] P. N. Mendes, H. Mühleisen, and C. Bizer. Sieve: Linked Data Quality Assessment and Fusion. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 116–123. ACM, 2012.
- [8] J. Michelfeit and T. Knap. Linked Data Fusion in ODCleanStore. In *International Semantic Web Conference (Posters & Demos)*, 2012.
- [9] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a Document-Oriented Lookup Index for Open Linked Data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52, 2008.
- [10] G. G. Shanks and P. Darke. Understanding Data Quality and Data Warehousing: A Semiotic Approach. In *Third Conference on Information Quality (IQ’98)*, pages 292–309, 1998.
- [11] E. Tsiflidou and N. Manouselis. Tools and Techniques for Assessing Metadata Quality. In *7th Metadata and Semantics Research Conference (MTSR’13)*, 2013.
- [12] Y. Tzitzikas, C. Alloca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating Heterogeneous and Distributed Information about Marine Species through a Top Level Ontology. In *Proceedings of the 7th Metadata and Semantics Research Conference (MTSR’13)*, Thessaloniki, Greece, November 2013.
- [13] Y. Tzitzikas, M. Kampouraki, and A. Analyti. Curating the Specificity of Ontological Descriptions under Ontology Evolution. *Journal on Data Semantics*, pages 1–32, 2013.
- [14] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk - A Link Discovery Framework for the Web of Data. In *Proceedings of the WWW’09 Workshop on Linked Data on the Web*, 2009.

## APPENDIX

### A. COMPETENCY QUERIES

Figure 4 displays the textual description for some competency queries as they were supplied by the communities.

#Query	For a scientific name of a species (e.g. <i>Thunnus Albacares</i> or <i>Poromitra Crassiceps</i> ), find/give me
Q <sub>1</sub>	the biological environments (e.g. <b>ecosystems</b> ) in which the <b>species</b> has been <b>introduced</b> and more general descriptive information of it (such as the <b>country</b> )
Q <sub>2</sub>	its <b>common names</b> and their complementary info (e.g. <b>languages</b> and <b>countries</b> where they are used)
Q <sub>3</sub>	the <b>water areas</b> and their <b>FAO codes</b> in which the <b>species</b> is <b>native</b>
Q <sub>4</sub>	the <b>countries</b> in which the <b>species</b> <b>lives</b>
Q <sub>5</sub>	the <b>water areas</b> and the <b>FAO</b> portioning <b>code</b> associated with a <b>country</b>
Q <sub>6</sub>	the presentation w.r.t. <b>Country</b> , <b>Ecosystem</b> , <b>Water Area</b> and <b>Exclusive Economic Zone</b> (of the water area)
Q <sub>7</sub>	the projection w.r.t. <b>Ecosystem</b> and <b>Competitor</b> , providing for each competitor the <b>identification information</b> (e.g. several codes provided by different organizations)
Q <sub>8</sub>	a map w.r.t. <b>Country</b> and <b>Predator</b> , providing for each predator both the <b>identification information</b> and the <b>biological classification</b>
Q <sub>9</sub>	who discovered it, in which <b>year</b> , the <b>biological classification</b> , the <b>identification information</b> , the <b>common names</b> - providing for each common name the <b>language</b> , the <b>countries</b> where it is used in.

Figure 4: Some indicative competency queries

# Scalable Numerical SPARQL Queries over Relational Databases

Minpeng Zhu, Silvia Stefanova, Thanh Truong, Tore Risch

Department of Information Technology, Uppsala University

Box 337, SE-75105 Uppsala, Sweden

{Minpeng.Zhu, Silvia.Stefanova, Thanh.Truong, Tore.Risch}@it.uu.se

## ABSTRACT

We present an approach for scalable processing of SPARQL queries to RDF views of numerical data stored in relational databases (RDBs). Such queries include numerical expressions, inequalities, comparisons, etc. inside FILTERs. We call such FILTERs *numerical expressions* and the queries - *numerical SPARQL queries*. For scalable execution of numerical SPARQL queries over RDBs, numerical operators should be pushed into SQL rather than executing the filters as post-processing outside the RDB; otherwise the query execution is slowed down, since a lot of data is transported from the RDB server and furthermore indexes on the server are not utilized. The *NUMTranslator* algorithm converts numerical expressions in numerical SPARQL queries into corresponding SQL expressions. We show that NUMTranslator improves substantially the scalability of SPARQL queries based on a benchmark that analyses numerical logs stored in an RDB. We compared the performance of our approach with the performance of other systems processing SPARQL queries to RDF views of RDBs and show that NUMTranslator improves substantially the scalability of numerical queries compared to the other systems' approaches.

## Keywords

SPARQL queries; RDF views of relational databases; numerical expressions; query rewrites; query optimization

## 1. INTRODUCTION

The Semantic Web provides uniform data representation for integrating data from different data sources by using established well-known formats like RDF, RDFS, OWL, and the standard query language SPARQL. Semantic Web seems promising to integrate and search industrial data [2].

Our application scenario is from the industrial domain, where sensors on machines such as trucks, pumps, kilns, etc., produce large volumes of log data. Such log data describes measured values of certain components at different times and can be used for analyzing machine behavior. Furthermore, the geographic locations of machines are often widely distributed and maintained locally in autonomous RDBs called *log databases*. We are developing the FLOQ (Federated LOG database Query) system, which is a system for historical analyses over federations of autonomous log databases using SPARQL

queries. To discover abnormal machine behaviors, a user of FLOQ defines SPARQL queries to these log databases. FLOQ processes a SPARQL query by first finding the relevant log databases containing the desired data, then sending local SPARQL queries to them, and finally collecting the local query results to obtain the final result.

In this paper we concentrate on scalable historical analyses by SPARQL queries of log data stored in a single relational database. Suspected abnormal machine behaviors are discovered and analyzed by specifying numerical SPARQL queries to an RDF view of the RDB. The queries analyze log data through numerical FILTERs containing numerical operators [11]. For example, query *Q1* retrieves the machine identifiers *m* for which a sensor has measured values *mv* of measurement class *A* higher than the expected values *ev* by a threshold value *@thA* during the time from *bt* to time *et*. Here *<prod>* denotes the URI for the RDF view of the RDB.

### Q1 :

```
SELECT ?m ?bt ?et
FROM <prod>
WHERE {?measuresA log:mA_BySensor ?sensor.
      ?measuresA log:mA/bt ?bt.
      ?measuresA log:mA/et ?et.
      ?measuresA log:mA/m ?m.
      ?measuresA log:mA/mv ?mv.
      ?sensor log:sensor/ev ?ev.
      FILTER (?mv > (?ev + @thA)) }
```

In FLOQ, SPARQL queries to RDBs are processed by generating a local execution plan containing calls to one or several SQL queries sent to a back-end RDBMS for evaluation. SPARQL queries that cannot be completely processed by SQL are instead partially processed by an execution plan interpreter in FLOQ. However, in order for the SQL queries to return the minimal required data, it is desirable that as much as possible of the SPARQL query is translated to SQL [8].

In FLOQ numerical SPARQL queries are defined over an automatically generated RDF view over an RDB expressed in *ObjectLog* [6], which is a Datalog dialect that supports objects for representing URIs and typed literals [9], disjunctive queries for UNION expressions, and foreign predicates to represent numerical operators in queries. The SPARQL queries are parsed into ObjectLog queries to the RDF view. Internally representing queries in ObjectLog permits domain calculus query transformations and optimizations before generating the execution plan. Calls to tuple calculus SQL query strings are made as foreign predicates. Foreign predicates are also used for accessing URIs in the execution plan. Doing all processing in

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.



Next we define two more typical numerical SPARQL queries to the log database,  $Q_2$  and  $Q_3$ , that discover abnormal machine behaviors. Query  $Q_2$  identifies a potential failure by retrieving for machine models  $M_1$ ,  $M_2$ , and  $M_3$  those *machineid* where, during the time interval ( $bt$ ,  $et$ ), the measured value  $mv$  was above 75% of the allowed deviation  $@thA$  from the expected value  $ev$ .

```

Q2:
SELECT ?machineid ?bt ?et
FROM <prod>
WHERE{?measuresA log:mA_bySensor ?sensor.
?measuresA log:mA/bt ?bt.
?measuresA log:mA/et ?et.
?measuresA log:mA/mv ?mv.
?measuresA log:mA_atMachine ?machineid.
?machineid log:machine/mm ?mm.
FILTER (?mm in ('M_1','M_2','M_3')).
?sensor log:sensor/ev ?ev.
FILTER (?mv > (?ev + 0.75*@thA)) }

```

Query  $Q_3$  identifies abnormal behaviors of machines of a measurement class based on absolute deviations: when and for which machine identifiers did the pressure reading of class  $B$  deviate more than  $@thB$  from its expected value  $ev$ ?

```

Q3:
SELECT ?m ?bt ?et
FROM <prod>
WHERE {?measuresB log:mB/bt ?bt.
?measuresB log:mB/et ?et.
?measuresB log:mB/mv ?mv.
?measuresB log:mB_bySensor ?sensor.
?sensor log:sensor/m ?m.
?sensor log:sensor/ev ?ev.
BIND ((?mv-?ev) as ?temp).
FILTER (abs(?temp) > @thB) }

```

### 3. FLOQ OVERVIEW AND QUERY PROCESSING

Figure 3 illustrates processing of numerical SPARQL queries by FLOQ.

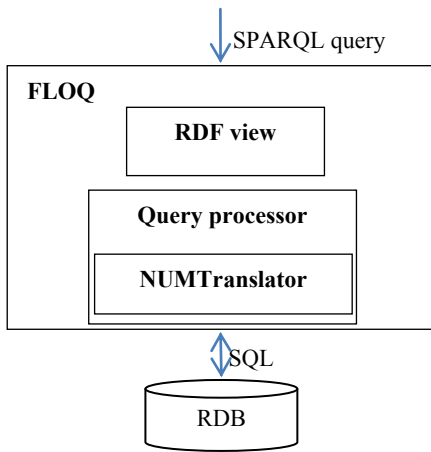


Figure 3. FLOQ query processor

The *RDF view* over the RDB is automatically generated based on the database schema and ontology mapping tables in FLOQ.

The used mappings conform to the direct mapping recommended by W3C [10].

We define a unique RDFS class for each relational table, except for link tables [10] representing set-valued properties as many-to-many relationships. In addition, RDF properties are defined for each column in a table. For example, the RDFS class with the URI  $\langle log:mA \rangle$  represents the table *MeasuresA*, while  $\langle log:mA/bt \rangle$  and  $\langle log:mA/et \rangle$  represent the columns *bt* and *et* in *MeasuresA*, respectively.

The RDF view is defined in terms of:

- *Source predicates*  $R(a_1, a_2, \dots, a_n)$  that represent the content of each referenced relational database table  $R$  where the tuple  $(a_1, \dots, a_n)$  represents a row in  $R$ .
- *URI-constructor* predicates that construct URIs to identify rows in tables.
- *Mapping tables* that map relational schema elements to RDF concepts.

The complete RDF view definitions can be found in [9]. The query processing steps in FLOQ are shown in Figure 4.

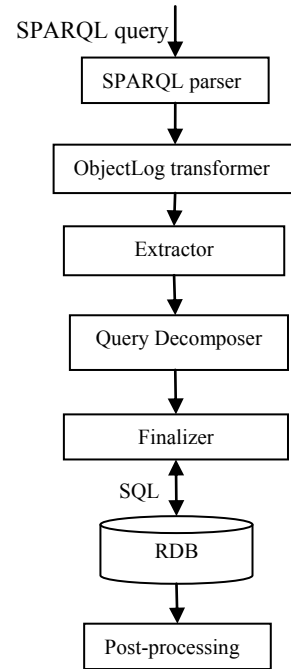


Figure 4. Query processing steps

The *SPARQL parser* first transforms the SPARQL query into an *ObjectLog* expression where each triple pattern in the query becomes a reference to the RDF view of the RDB. Then the *ObjectLog transformer* generates a simplified disjunctive normal form (DNF) predicate. The *NUMTranslator* algorithm performs the *extractor* and *finalizer* steps. The extractor collects from conjunctions predicates that can be translated to SQL, called *access filters*. The *query decomposer* then optimizes the query, producing a query execution plan where access filters are called. The finalizer traverses the execution plan to translate the extracted predicates in the access filters into SQL expressions. When the execution plan is interpreted, the generated SQL statements are sent to the RDB for execution. The non-extracted predicates are not translated to SQL and have to be processed outside the RDB by *post-processing* operators. For example,

such operators are URI-constructors and numerical expressions not supported by the SQL engine.

## 4. THE NUMTRANSLATOR ALGORITHM

The NUMTranslator uses a table-driven approach to define which SPARQL operators to extract and translate into corresponding SQL operators and functions. Table 1 defines the SPARQL to SQL operator translations:

**Table 1. SPARQL to SQL operators to translate**

SPARQL	SQL	INFIX	FUNCTION
>	>	True	False
<	<	True	False
=	=	True	False
!=	<>	True	False
+	+	True	True
-	-	True	True
ABS	ABS	False	True
UCASE	UPPER	False	True
etc.			

In Table 1 there is one row for each SPARQL operator or function (column *SPARQL*) that can be translated into SQL. The column *SQL* defines the corresponding SQL operator or function. A value in the column *INFIX* is true when the corresponding SQL operator is an infix operator *op* on operands *x* and *y*, i.e.  $x \text{ op } y$  (e.g.  $x+y$ ); otherwise it is an SQL function on format  $f(x,y,...)$ . The column *FUNCTION* is true when the operator is a non-Boolean function returning a value.

### 4.1 The NUMTranslator extractor

The extractor is applied on each ObjectLog conjunction in the simplified predicate received by the ObjectLog transformer. The extractor collects predicates that can be translated to SQL. Such predicates are i) source predicates *SPs* representing RDB tables, and ii) *non-source predicates (NSPs)* that are defined in Table 1 as translatable to SQL.

Figure 5 shows the ObjectLog representation of *Q1* after it has been transformed by the ObjectLog transformer.

```
Q1(m, bt, et):-
1 MeasuresA(m, s, bt, et, mv)      and
2 mv > v36                        and
3 v36 = ev + @thA                 and
4 Sensor(m, s, _, _, ev, _, _)
```

**Figure 5. ObjectLog of query Q1**

In this case all predicates in *Q1* are translatable to SQL since *MeasuresA* and *Sensor* are SPs, and *>* and *+* are NSPs defined in Table 1.

The steps of the extractor are the following:

1. Initialize a variable *Xpreds* for the first found SP, denoted *R1*, in the conjunction and bind a variable *Rest* to the other predicates.
2. Iteratively extract from *Rest* the predicates that have some common variable with some extracted predicate in *Xpreds*, which are either SPs or NSPs defined in Table 1.
3. Construct an *access filter* of all extracted predicates in *Xpreds* since those can be fully translated to SQL.

4. While there are some remaining SP, *R2*, in *Rest*, re-initialize *Xpreds* by *R2* and *Rest* by the remaining predicates, and repeat steps 2-3.
5. Finally, construct a conjunction of the access filters and *Rest*.

For example, for *Q1* the predicates in *Xpreds* are extracted in the following order:

1. *MeasuresA(m, s, bt, et, mv)* (line 1), since it is an SP.
2. *>(mv, v36)* (line 2) since *>* is defined in Table 1 and the variable *mv* is common with the extracted *MeasuresA*.
3. *Sensor(m, s, \_, \_, ev, \_, \_)* (line 4) since it is an SP having common variables (*m* and *s*) with *MeasuresA()*.
4. *v36 = ev + @thA* (line 3) since *+* is defined in Table 1 and the variable *ev* is common with the extracted *Sensor* predicate.

Then the following conjunctive access filter *F1* is formed by the predicates in *Xpreds*:

```
F1(m, s, bt, et, mv, ev) :-
1 MeasuresA(m, s, bt, et, mv)      and
2 Sensor(m, s, _, _, ev, _, _)    and
3 v36 = ev + @thA                 and
4 mv > v36
```

No non-translatable predicates remain in *Rest*.

### 4.2 Query decomposition

To optimize the query produced by the extractor, the query decomposer uses cost-based optimization [6] to produce an optimized execution plan. Based on heuristics and statistic of the queried RDB, execution cost and selectivities of access filter are estimated. Default cost parameters are used by the optimizer to estimate the execution cost and selectivities of predicates if no statistic is available. The decomposer will then reorder the access filters and the post processed predicates to generate an optimized execution plan. We do not further elaborate the query decomposer here.

### 4.3 The NUMTranslator finalizer

The finalizer translates access filters in the decomposed execution plan into calls to an SQL interface operator, *sql* that sends generated SQL strings to the back-end RDB for execution.

ObjectLog numerical expressions are translated into SQL numerical expressions by recursively replacing all ObjectLog domain variables that represent numerical expressions with their bound expressions. For example, the variable *v36* in line 4 in *F1* doesn't represent a relational column and is replaced by its bound expression in line 3, and then the obtained expressions is  $mv > ev + @thA$ . Thus for *Q1* the execution plan *P1* becomes the following:

```
(m, bt, et)
↑
γ sql(ds, "SELECT m.m, bt, et FROM MeasuresA
m, SENSOR s WHERE m.mv > s.ev + @thA AND
m.m=s.m AND m.s=s.s", (m, bt, et))
```

**Figure 6. Execution plan P1 with NUMTranslator**

The execution plan contains an algebra expression where the *apply* operator  $\gamma \text{ fn}(\cdot)$  calls the *foreign predicate* *sql(ds, q, result)* implemented in Java. The foreign predicate *sql* sends an

SQL query  $q$  to the RDBMS data source  $ds$  for execution and iteratively returns bindings of tuples,  $result$ .

If NUMTranslator had not been applied, all numerical operators would have to be post-processed, which would slow down the query execution since filtering cannot be made in the database server.

For example, if NUMTranslator is turned off, for  $Q1$  the following execution plan  $P2$  is produced that doesn't contain any numerical SQL operators corresponding to numerical SPARQL operators, which are instead post-processed:

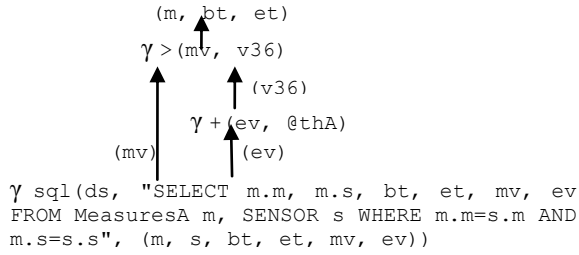


Figure 7. Execution plan  $P2$  without NUMTranslator

Comparing the two execution plans  $P1$  and  $P2$  it can be seen that the  $sql$  operator in  $P2$  retrieves much more data than  $P1$ , so if NUMTranslator is turned off lots of data needs to be filtered out outside the RDB server. Furthermore, the utilization of indexes on the SQL numerical expression by the back-end database server makes significant performance difference. We show in the next section that applying NUMTranslator substantially improves the query performance of numerical SPARQL queries.

## 5. PERFORMANCE MEASUREMENTS

We compared the performance for executing the numerical queries  $Q1$ ,  $Q2$ , and  $Q3$  in FLOQ with and without applying NUMTranslator. Furthermore, we compared the query performance of FLOQ with the query performance of D2RQ [1] for  $Q1$ ,  $Q2$ , and  $Q3$ , for the same back-end relational database. We tried to run the queries with both ontop [7] and Virtuoso [3] as well, but none of our numerical SPARQL queries could be run, indicating that those systems do not provide full support for processing numerical SPARQL queries.

All experiments are carried out on a MS SQL Server 2008 R2 installed on a server machine with 8 AMD Opteron™ 6128 processors, 2.00 GHz CPU and 16GB RAM. The RDB is populated by loading sensor data into the MS SQL server. B-tree indexes are created on the columns  $mm$ ,  $mv$ ,  $bt$ ,  $et$ ,  $ev$ ,  $ad$ , and  $rd$  to speed up the queries.

All measurements were taken both for cold and warm runs. The cold runs were made immediately after the RDBMS server was started, which implied that there were no data cached in the buffer pool and the executed query wasn't optimized by the RDBMS. Thus a measured query execution time for a cold run includes the time for i) reading data from disk, ii) SQL query optimization on the RDBMS server, iii) communication, and iv) post-processing of data on the client. The warm runs were made after a query was executed once. Since the back-end RDBMS has a statement cache a same SQL query executed twice will be optimized the first time it is run. Therefore, warm executions do not include RDBMS query optimization time.

The plotted values are mean values of three measurements. The standard deviation is less than 10% in all cases. To investigate the SQL query produced by all the other systems we use the system profiling tool of MS SQL server when running a query.

The following notations are used in the performance diagrams:

- *NUMTranslator*: FLOQ with NUMTranslator turned on, i.e. the SPARQL numerical expressions are translated into corresponding SQL expressions.
- *Naive*: FLOQ with NUMTranslator turned off, i.e. the SPARQL numerical expressions are not translated into corresponding SQL numerical expressions.
- *D2RQ*: D2RQ version [0.8.1] configured with the system's default mappings.

Figure 8, 9 and 10 show the execution times for both cold and warm runs for  $Q1$ ,  $Q3$ , and  $Q2$  while scaling the databases size from 1 GB to 15 GB.

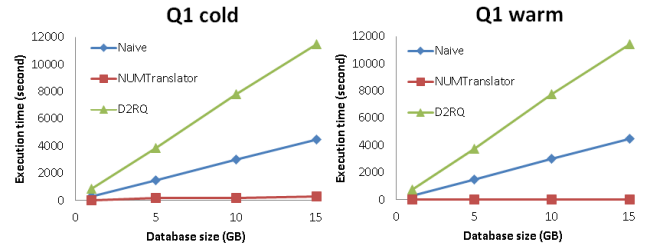


Figure 8. Execution times for  $Q1$

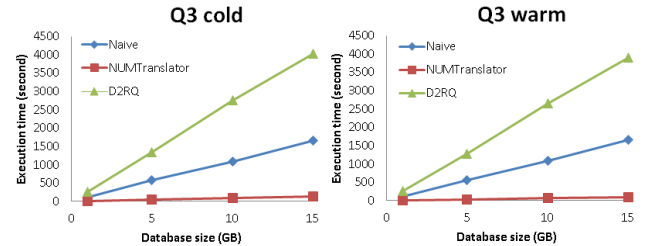


Figure 9. Execution times for  $Q3$

Figure 8 and 9 show that *NUMTranslator* substantially improves the query execution scalability compared to *Naive* for numerical SPARQL queries like  $Q1$  and  $Q3$  with highly selective numerical FILTERs: 0.04% for  $Q1$  and 3% for  $Q3$ . In these cases pushing the numerical FILTERs to SQL is more profitable than filtering large data amounts on the client. The performance of D2RQ is worse than *Naive* since D2RQ sends to the RDBMS an SQL query that doesn't contain numerical expressions, and is a much more complex query with more joins. Furthermore,  $Q3$  had to be manually changed for D2RQ to remove the BIND operator, since otherwise D2RQ wouldn't return correct result.

Measurement results for  $Q2$  are shown in Figure 10. For  $Q2$  the results for *NUMTranslator* and *Naive* are presented in a separate diagram, since they are very close. It can be seen on Figure 10 that *NUMTranslator* doesn't improve the query performance for non-selective queries like  $Q2$  where the FILTER selects 43% of the data. In this case pushing the numerical SPARQL filters to be executed to the RDBMS server doesn't make a significant difference compared to post-filtering data on the client.

D2RQ performs worse for  $Q2$  since it doesn't translate any of the FILTERs and it furthermore generates a very complex SQL query with many joins.



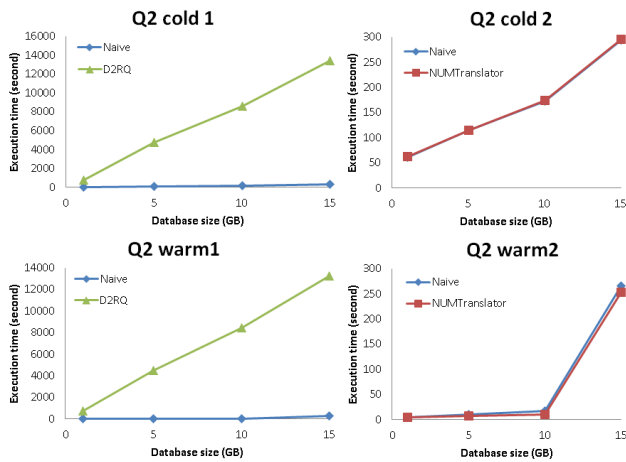


Figure 10. Execution times for  $Q_2$

In general, the experiments show that NUMTranslator substantially improves the query performance of numerical SPARQL queries where the numerical FILTERs have high selectivity.

## 6. RELATED WORK

Virtuoso RDF Views [3] and D2RQ [1] are other systems that process SPARQL queries to RDF views of RDBs. These systems implement compilers that translate SPARQL directly to SQL. By contrast, FLOQ first generates ObjectLog queries to a declarative RDF view of the RDB, and then transforms the SPARQL queries to SQL by logical transformations.

We didn't find any publication of how D2RQ compiles numerical SPARQL queries into SQL and the documentation for Virtuoso's SQL generation is very limited [3]. However, by using the profiling tool of the RDBMS and the debug logging of Virtuoso we were able to analyze what queries were actually sent to the RDBMS, showing that neither of those systems translates numerical SPARQL expressions into corresponding SQL expressions.

The ontop system [7] also enables SPARQL queries to RDF views of RDBs by translating SPARQL to Datalog programs, which are rewritten and translated to SQL. A difference to ontop is the table driven NUMTranslator algorithm, which makes it very easy to extend for new operators. Furthermore, FLOQ generates execution plans containing calls to SQL intermixed with expressions interpreted in the client. This enables FLOQ to interpret in the client SPARQL operators not available in SQL. In addition NUMTranslator translates the domain calculus SPARQL queries into tuple calculus SQL queries by substituting variables with their bound expressions.

## 7. CONCLUSIONS AND FUTURE WORK

We presented the FLOQ system where the NUMTranslator algorithm uses a table driven approach to translate numerical domain calculus SPARQL expressions into corresponding numerical SQL expressions. This enables scalable processing of numerical SPARQL queries to RDF views over RDBs.

The approach was evaluated on a benchmark scenario in an industrial setting where logged data stored in an RDB was analyzed using numerical SPARQL queries. We compared the performance of the SPARQL queries with and without applying NUMTranslator. The experiments show that NUMTranslator substantially improves the query performance of numerical

SPARQL queries in particular when the numerical expressions inside FILTERs are highly selective.

We also compared our approach with other systems that translate SPARQL queries to SQL. Only D2RQ could execute our queries, but substantially slower since D2RQ does not employ an approach similar to NUMTranslator.

As our next step, we will investigate numerical SPARQL queries searching large numbers of distributed log databases combined through an ontology. Another issue is creating benchmarks based on randomly generating SPARQL queries [5]. Furthermore, query processing and mediation strategies over other back-ends than RDBs [4] in our setting should be investigated.

## 8. ACKNOWLEDGMENTS

This work is supported by EU FP7 project Smart Vortex and the Swedish Foundation for Strategic Research under contract RIT08-0041.

## 9. REFERENCES

- [1] Bizer, C., Cyganiak, R., Garbers, G., Maresch, O., and Becker, C. 2009. *The D2RQ Platform v0.7 - Treating Non-RDF Relational Databases as Virtual RDF Graph*, <http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/>
- [2] Björkelund, A., Edström, L., etc. 2011. On the integration of skilled robot motions for productivity in manufacturing, In *Proc. of IEEE International Symposium on Assembly and Manufacturing*, Tampere, Finland.
- [3] Erling, O. and Mikhailov, I. 2009. RDF Support in the Virtuoso DBMS, *Studies in Computational Intelligence*, Vol. 221
- [4] Langegger, A., Wöb, W., and Blöchl, M. 2008. A Semantic Web Middleware for Virtual Data Integration on the Web, *5th European Semantic Web Conference ESWC 2008*.
- [5] Langegger, A. and Wöb, W. 2009. RDFStats – The Extensible RDF Statistics Generator and Library, *8th International Workshop on Web Semantics, DEXA 2009*, Linz, Austria, August 31-September 40.
- [6] Litwin, W. and Risch, T. 1992. Main Memory Oriented Optimization of OO Queries using Typed Datalog with Foreign Predicates, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 6.
- [7] Rodriguez-Muro, M., Rezk, M., Hardi, J., Slusnys, M., Bagoši, T., and Calvanese, D. 2013. Evaluating SPARQL-to-SQL Translation in Ontop, *ORE 2013*
- [8] Sequeda, J. F., and Miranker, D. P. 2013. *Ultrawrap: SPARQL Execution on Relational Data*, Tech. Report, Univ. of Texas at Austin. [http://apps.cs.utexas.edu/tech\\_reports/reports/tr/TR-2078.pdf](http://apps.cs.utexas.edu/tech_reports/reports/tr/TR-2078.pdf)
- [9] Stefanova, S., and Risch, T. 2011. Optimizing Unbound-property Queries to RDF Views of Relational Databases. *7th International workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011)*, Bonn, Germany.
- [10] Arenas, M., Bertails, A., Prud'hommeaux, E., and Sequeda, J. 2012. A Direct Mapping of Relational Data to RDF, <http://www.w3.org/TR/rdb-direct-mapping/>
- [11] Harris, S., and Seaborne, A. 2013. SPARQL 1.1 Query Language, <http://www.w3.org/TR/sparql11-query/>



# Similarity Recognition in the Web of Data

Alfio Ferrara  
Dipartimento di Informatica  
Università degli Studi di Milano  
Via Comelico 39  
20135 - Milano, Italy  
alfio.ferrara@unimi.it

Lorenzo Genta  
Dipartimento di Informatica  
Università degli Studi di Milano  
Via Comelico 39  
20135 - Milano, Italy  
lorenzo.genta@unimi.it

Stefano Montanelli  
Dipartimento di Informatica  
Università degli Studi di Milano  
Via Comelico 39  
20135 - Milano, Italy  
stefano.montanelli@unimi.it

## ABSTRACT

In the web of data, similarity recognition is the basis for a variety of resource-consuming activities and applications, including data recommendation, data aggregation, and data analysis. In this paper, we propose HMatch4, a novel instance matching algorithm for similarity recognition, which has been developed on the ground of our experience with HMatch3 [3].

## 1. INTRODUCTION

In the web of data, the capability to recognize the degree of similarity between different descriptions of web resources is getting more and more crucial for a number of purposes [2, 5]. Focused techniques specifically conceived for the web of data are required to address the peculiar aspects of similarity evaluation in such a context. Two main issues need to be considered.

**Similarity is not identity.** In the literature, traditional approaches/tools for similarity recognition are based on the idea of comparing resources by analyzing their features. In these solutions, the goal is to recognize the *identity* between two resources, namely the capability to detect when the two descriptions refer to the same real object. From this perspective, identity is seen as a special case of similarity characterized by a high similarity value (i.e., high number of shared features). Low similarity values are usually interpreted as a *non-identity* result, meaning that the two considered resource descriptions refer to different real objects. Consider the following example:

**tiger woods**  
profession: golfer  
nationality: united states

**arnold schwarzenegger**  
profession: bodybuilder  
profession: politician  
nationality: united states

In this case, the degree of similarity between **tiger woods** and **arnold schwarzenegger** is quite low, because they ac-

tually have only one feature in common (i.e., nationality). According to traditional instance matching approaches, the similarity value between the two resources is discarded, concluding that they do not represent the same person. Such a behavior is correct, however, matching techniques for the web of data have to be capable of capturing and preserving the correct degree of similarity both when the considered resources are identical or very similar and when they are only quite similar or even completely different. This can be useful in many application scenarios, like for example in web data classification where the goal can be to aggregate resources based on the similarity over one (or few) specific features (see for example [4]).

**The scale as a key issue.** In the web of data, matching usually involves very large data collections, potentially composed by hundred of thousands of web resources described by millions of features. In this scenario, the intrinsic limitation of existing instance matching approaches to similarity evaluation is due to the cost of directly comparing all the pairs of resource items in order to compute their degree of similarity within a considered dataset. In a classical *comparison-based* matching approach, this means that a matching operation is required to calculate the similarity degree for each possible pair of items. For a dataset of  $n$  items, this means that  $O(n^2)$  matching operations are required in the worst case to compare each item against all the other items in the dataset. Optimization strategies have been proposed in the literature to reduce the number of comparison operations and to increase the performance of the overall matching process [5]. However, to the best of our knowledge, the currently available instance matching tools are still affected by severe limitations in terms of scalability, which usually means that in real systems the similarity recognition task need to be executed offline in case of very large collections of data.

In this paper, we propose HMatch4, a novel instance matching algorithm for similarity recognition, which has been developed on the ground of our experience with HMatch3 [3]. HMatch4 is natively conceived for working in the web of data, where the above matching issues are properly considered and addressed. In the following, we first describe the HMatch4 techniques and related algorithm (Section 2 and 3). Then, we provide the results of a preliminary evaluation obtained by comparing HMatch4 against HMatch3 and other popular tools for instance matching, namely LogMap [7] and SLINT+ [8] (Section 4). Related work and concluding re-

marks are finally discussed (Section 5 and 6).

## 2. THE HMATCH4 PROCESS

We first describe our model for representation of the web-of-data resources, and then we present the HMatch4 matching process.

### 2.1 Modeling web resource items

In HMatch4, we rely on the use of an internal data model called *web resource item (wri)* for representation of the resources to match. A wri element is featured by a set of feature-value pairs and it is defined as follows:

$$wri = \{(f, v)_1 \dots (f, v)_k\}$$

where each pair  $(f, v) \in wri$  represents a feature-name  $f$  and the corresponding feature-value  $v$ .

**Wrapping resources to the wri model.** The wri model has been conceived to support matching of different kinds of web-of-data resources, like for example social data (e.g., Facebook, Twitter resources) and linked data (e.g., Freebase, DBpedia resources). The idea of modeling resources as sets of feature-value pairs is motivated by the need to deal with a number of different native formats that are commonly employed for description of web-of-data resources. Appropriate wrapping operations are required to transform the native web resource representation into a wri-based representation. In general, this wrapping step is straightforward. A feature-value pair  $(f, v) \in wri$  is created for each property and corresponding value within the native description of the considered resource. For instance, in case of a social data resource like a tweet, a feature-value pair is defined in the wri representation for each tweet field (e.g., id, text, user, lang, place, created\_at, entities). A similar approach is enforced to generate a wri representation when a linked data resource *URI* extracted from a repository  $\mathcal{R}$  is considered. In particular, a feature-value pair is created for each property name and corresponding property value that is directly connected with *URI* in the RDF specification extracted from the repository  $\mathcal{R}$ . In case that *URI* has a property name  $p$  associated with multiple property values  $v_1 \dots v_m$ , a feature-value pair  $(p, v_j)$  is created in the wri description for each value  $v_j$  with  $j \in [1, m]$ .

**Example.** As an example, we show the wri description for a linked data resource featuring the famous athlete muhammad ali. Such a description is extracted from the Freebase repository by only considering the properties `profession`, `type`, and `nationality`.

```

muhammad_ali
-----
(profession, athlete),
(profession, professional boxer),
(type, olympic athlete),
(nationality, United States of America).

```

### 2.2 Matching process

HMatch4 works on a dataset  $\mathcal{D}$  of wri elements to match and it produces a similarity matrix as a result.

**Spirit of HMatch4.** The idea is to measure the similarity degree between two items by calculating their number

of common feature-value pairs in the wri representations. Given  $wri_1$  and  $wri_2$ , this can be determined by calculating the set of pairs  $(f, v)$  that belong to both the considered items (i.e.,  $wri_1 \cap wri_2$ ). As a difference with classical comparison-based approaches, HMatch4 proposes a sort of *index-based* matching approach where the similarity degree of two items is the result of an indexing operation and a “pair-by-pair” comparison between wri elements is not required. The key idea of HMatch4 is to consider each single  $wri$  in the dataset and to index all the possible subsets of feature-value pairs belonging to  $wri$  that can be relevant for detecting a similarity with other wri elements. Two items  $wri_1$  and  $wri_2$  are similar if they share the same entry in the index, meaning that they have a common subset of feature-value pairs in their wri representations (i.e., the feature value pairs of the index entry). The similarity degree is assessed by measuring the size (i.e., cardinality) of the shared subset of feature-value pairs.

**Matching process.** The matching process of HMatch4 is articulated in three main steps, namely *configuration*, *execution*, and *assessment* (see Figure 1). The **configuration step** defines the setup of the matching execution and it specifies the requirements to be satisfied by two items  $wri_1$  and  $wri_2$  for being considered as similar. We call *feature-set*  $F$  the set of all the features involved in the specification of wri elements within the considered dataset  $\mathcal{D}$ , namely:

$$F = \left\{ \bigcup_{i=1}^n fs(wri_i) \right\}$$

where  $n = |\mathcal{D}|$  is the number of items within the dataset  $\mathcal{D}$  and  $fs(wri) = \{f_j \mid (f_j, v) \in wri\}$  is the set of features characterizing the feature-value pairs of  $wri$ . Then, we calculate the power set  $\mathcal{P}(F)$  containing all the possible subsets of features over  $F$ . Then, we define the set  $\mathcal{F} \subset \mathcal{P}(F)$  as follows:

$$\mathcal{F} = \left\{ rfs \mid rfs \in \mathcal{P}(F) \wedge \frac{|rfs|}{|F|} \geq th_s \right\}$$

where  $th_s \in (0, 1]$  is a similarity threshold and it determines the minimum similarity degree that is required to consider two items as matching items. A set  $rfs \in \mathcal{F}$  is called *relevant feature-set* and it represents a combination of features to be considered for similarity recognition. The rationale of our matching process is that two items  $wri_1$  and  $wri_2$  are similar iff they share a set of feature-value pairs where the features coincides with a set  $rfs \in \mathcal{F}$ .

The **execution step** creates an index structure  $\mathcal{I}$  containing entries for the combinations of feature-value pairs within wri descriptions that are relevant for similarity recognition. An index entry  $ie \in \mathcal{I}$  has the form  $ie = \langle rfs, fvl, wp \rangle$ , where  $rfs \in \mathcal{F}$  is a relevant feature-set,  $fvl$  is a list of feature-value pairs, and  $wp$  is a set of wri elements belonging to  $\mathcal{D}$ . Given an index entry  $ie$  and the associated relevant feature-set  $rfs = \{f_1, \dots, f_s\}$ , the corresponding list of feature-value pairs  $fvl$  has the form  $fvl = (f_1, v_1) \mid \dots \mid (f_s, v_s)$ , and  $wp$  contains the wri elements that provide  $fvl$  in their wri representation. For each item  $wri \in \mathcal{D}$ , we create an entry in the index  $\mathcal{I}$  for those combinations of feature-value pairs of  $wri$  that are based on a relevant feature-set belonging to  $\mathcal{F}$ . Details about the HMatch4 algorithm for creating the index  $\mathcal{I}$  are presented in Section 3.

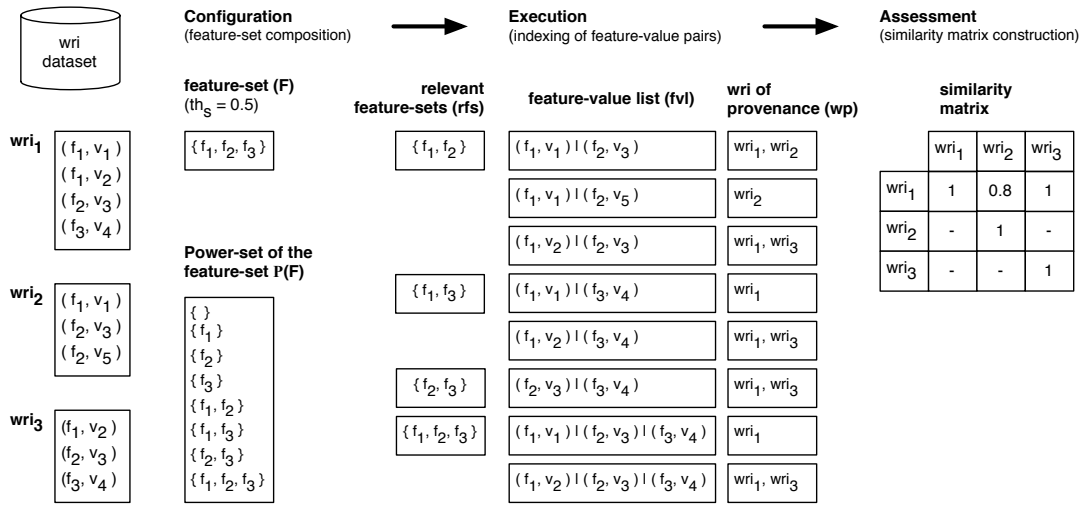


Figure 1: The similarity recognition process of HMatch4

The **assessment step** generates the similarity matrix  $\mathcal{M}$  for the wri items of the dataset  $\mathcal{D}$ . Given two items  $wri_1$  and  $wri_2$ , their similarity value  $sim(wri_1, wri_2)$  is determined by querying the index  $\mathcal{I}$  and by extracting the index entry shared by  $wri_1$  and  $wri_2$  (if it exists). Given an index entry  $ie$  where  $wri_1, wri_2 \in wp$ , the similarity value  $sim(wri_1, wri_2)$  is calculated through the Dice's coefficient formula:

$$sim(wri_1, wri_2) = \frac{2 \cdot |rfs|}{|fs(wri_1)| + |fs(wri_2)|}$$

where  $fs(wri)$  is the set of features characterizing the feature-value pairs of  $wri$  and  $rfs$  is the relevant feature-set associated with the index entry  $ie$ , meaning that the items  $wri_1$  and  $wri_2$  share feature-value pairs for all the features in  $rfs$ . According to this definition, the similarity values computed by HMatch4 are symmetric (i.e.,  $sim(wri_1, wri_2) = sim(wri_2, wri_1)$ ) and thus the resulting matrix  $\mathcal{M}$  is upper triangular. It is possible that two items  $wri_1$  and  $wri_2$  have not a shared entry in the index structure. In HMatch4, this means that  $sim(wri_1, wri_2) = 0$ . It is also possible that two or more index entries are shared by two items  $wri_1$  and  $wri_2$ , meaning that the two items have more relevant feature-sets in common. In this case, the entry with  $max(|rfs|)$  is selected for calculating  $sim(wri_1, wri_2)$ .

**Example.** We consider the wri about muhammad ali previously introduced and the following wri descriptions about michael\_jordan and george foreman.

<u>michael_jordan</u>
(profession, athlete), (type, olympic athlete), (type, celebrity).
<u>george_foreman</u>
(profession, professional boxer), (type, olympic athlete), (nationality, United States of America).

According to these descriptions, the feature-set  $F = \{ \text{pro-$

fession, type, nationality  $\}$  is generated. By setting a similarity threshold  $th_s = 0.5$ , we obtain the relevant feature-sets  $\mathcal{F} = \{ \{ \text{profession, type} \}, \{ \text{profession, nationality} \}, \{ \text{type, nationality} \}, \{ \text{profession, type, nationality} \} \}$ . The resulting index structure is shown in Figure 2. Thus, the similarity matrix generated in the assessment step is the following:

	m. ali	m. jordan	g. foreman
m. ali	1.0	0.8	1.0
m. jordan	-	1.0	0.0
g. foreman	-	-	1.0

For calculation of  $sim(m. ali, m. jordan)$ , we consider the first entry in the index structure of Figure 2 that is shared by m. ali and m. jordan (see column  $wp$ ). Thus, we obtain:  $sim(m. ali, m. jordan) = (2 \cdot 2) / (3 + 2) = 0.8$ . Moreover, we note that  $sim(m. jordan, g. foreman) = 0.0$ . This is due to the fact that m. jordan and g. foreman only share the feature-value pair  $\text{type, olympic athlete}$  that does not coincide with a relevant feature-set, meaning that the similarity between m. jordan and g. foreman is not sufficient for being recognized in the current HMatch4 configuration ( $th_s = 0.5$ ).

### 3. THE HMATCH4 ALGORITHM

In this section, we present the HMatch4 algorithm used in the execution step for creation of the index structure  $\mathcal{I}$ . The algorithm is shown in Figure 3. The algorithm is implemented by the function  $\text{INDEXING}(\mathcal{D}, \mathcal{F})$ , which takes the dataset  $\mathcal{D}$  and the relevant feature-set  $\mathcal{F}$  as input. The function initializes the index  $\mathcal{I}$  as a map where keys are numeric values and values are sets of  $wri$  (line 2). Then, it takes into account all the  $wri$  in  $\mathcal{D}$ . For each  $wri$  and for each relevant feature-set  $rfs$  in  $\mathcal{F}$ , we first initialize an empty set of values  $V$  (line 5). As an example, let us take into account the  $wri_1$  of Figure 1, which is composed by the feature-value pairs  $\{(f_1, v_1), (f_1, v_2), (f_2, v_3), (f_3, v_4)\}$ , and the relevant feature-set  $\{f_1, f_2\}$ . For each feature  $f_i$  in  $rfs$ , we insert into  $V$  all the feature-values pairs having  $f_i$  as feature (lines 7-8). In our example, the set  $V$  at the end of this process is composed by the elements  $V = \{ \{(f_1, v_1), (f_1, v_2)\},$

Relevant feature-set $rfs$	Feature-value list $fv_l$	wri $wp$
{profession, type}	(profession, athlete) (type, olympic athlete) (profession, athlete) (type, celebrity) (profession, professional boxer)   (type, olympic athlete)	m ali, m. jordan m. jordan
{profession, nationality}	(profession, athlete)   (nationality, United States of America) (profession, professional boxer)   (nationality, United States of America)	m. ali, g. foreman m ali
{type, nationality}	(type, olympic athlete)   (nationality, United States of America)	m. ali, , g. foreman
{profession, type, nationality}	(profession, athlete)   (type, olympic athlete)   (nationality, United States of America) (profession, professional boxer)   (type, olympic athlete)   (nationality, United States of America)	m.ali, g. foreman m ali m ali, g. foreman

Figure 2: Index structure for the wri descriptions about muhammad ali, micheal jordan, and george foreman

```

1: function INDEXING( $\mathcal{D}, \mathcal{F}$ )
2:    $\mathcal{I} \leftarrow$  a key-value map of the form <number: set>
3:   for all  $wri \in \mathcal{D}$  do
4:     for all  $rfs \in \mathcal{F}$  do
5:        $V \leftarrow \{\}$ 
6:       for all  $f \in rfs$  do
7:          $F \leftarrow \{(f_i, v_i) \mid (f_i, v_i) \in wri \wedge f_i = f\}$ 
8:         add  $F$  to  $V$ 
9:       end for
10:       $X \leftarrow v_1 \times v_2 \times \dots \times v_{|V|} \mid v_i \in V$ 
11:      for all  $x \in X$  do
12:         $h \leftarrow hash(x)$ 
13:        if  $h \in keys(\mathcal{I})$  then
14:          add  $wri$  to  $\mathcal{I}[h]$ 
15:        else
16:           $\mathcal{I}[h] \leftarrow \{wri\}$ 
17:        end if
18:      end for
19:    end for
20:  end for
21:  return  $\mathcal{I}$ 
22: end function

```

Figure 3: The HMatch4 execution algorithm

$\{(f_2, v_3)\}$ . Now, we process each element in the cartesian product  $X$  of the sets in  $V$ , which are  $\{(f_1, v_1), (f_2, v_3)\}$  and  $\{(f_1, v_2), (f_2, v_3)\}$  (lines 10-11). The idea behind this step is that features having more than one value, such as  $f_1$  in the example, are considered separately and in combination with all the other feature values. For each element  $x$  of  $X$ , we insert a new entry in the index  $\mathcal{I}$ . In particular, we obtain a numerical index key as the hash-value of  $x$ . We note that, in this paper, we assume to have just a simple hashing function capable of providing a unique value for each combination of feature-value pairs. The development of a more powerful hashing function is one of the goals of our future work. According to this procedure (lines 12-16), the index  $\mathcal{I}$  will contain one entry for each combination of values in the relevant feature-set  $rfs$ . In our example, given all the feature-sets of Figure 1, we generate 7 entries, 3 for  $\{f_1, f_2\}$ , 2 for  $\{f_1, f_3\}$ , 1 for  $\{f_2, f_3\}$ , and 2 for  $\{f_1, f_2, f_3\}$ . In such a way, when a subsequent  $wri_j$  is processed, if it has one or more combinations of feature-value pairs that are equal to those of  $wri$ , it will be inserted in the same set of  $wri$  in the index, denoting the fact that there is a similarity between  $wri_j$  and  $wri$ .

## 4. EXPERIMENTAL RESULTS

The goal of our experimentation is to evaluate HMatch4 in terms of i) the quality of similarity recognition measured by precision and recall; ii) the scalability of HMatch4 when matching a growing number of web resources. Considering the quality assessment, we performed a comparison against

a ground-truth set of mappings produced by human users. The scalability evaluation is performed on both time and space consumption in comparison with the matching tools LogMap and SLINT+ [7, 8]. These tools have been chosen for their known efficiency and for the availability of a working prototype. The scalability tests are performed on an automatically produced dataset based on multiple replications of a base dataset. Both the quality and scalability tests have been performed by comparing HMatch4 and HMatch3 [3], our previous version of matching tool.

### 4.1 Experiment setup

In this section, we discuss quality assessment and scalability evaluation.

**Quality assessment.** The ground truth has been produced by exploiting a novel crowdsourcing approach called *Liquid Crowd*. A crowdsourcing approach consists in reducing a problem in a set of elementary units of work that are distributed to a (possibly) large number of human workers. Each worker participates giving the solution for one or more work units and receives a reward (e.g., money, personal satisfaction or other benefits) proportional to the completed amount of work. The main idea behind Liquid Crowd is to change the definition of *worker* from a single user to a *group* of users. A work unit is considered *accepted* only if the assigned group reaches a consensus on the produced answer (i.e., the qualified majority of users converge on the same answer). In our experimentation, the ground truth for quality assessment is built on 58 individuals from Freebase repository with a total number of 275 feature-value pairs. Thus, the *work units* have been structured as a blind evaluation of a pair of web resources. For instance, the users have to evaluate the similarity of the given resources only knowing their features and features-values without knowing their identifiers (i.e., the names of the resources): this is done to avoid that users exploit their personal knowledge in evaluating the similarity.

In order to complete a work unit, the user has to choose between 4 possible answers: equal (E), very similar (VS), quite similar(QS), unequal (U). Thus, the final mappings produced are in the form  $m(wri_x, wri_y) = \{E|VS|QS|U\}$ . As a result from the proposed set of work units, the human workers produced 1136 mappings (work units that reached the consensus) with the following distribution:  $U = 653, QS = 317, VS = 151, E = 15$ . On this dataset we produced 1653 work units. The crowdsourcing session was open for 7 days to any volunteer: 82 persons took part to the experiment with a result of 1136 completed tasks.

**Scalability evaluation.** For this test, we performed differ-

ent executions of the 4 matching tools by replicating  $K$  times the dataset used for quality assessment, for  $K$  between 1 and 1461. The number of resources of the performed tests is between 58 (426 RDF triples) and 85144 (888293 RDF triples) and each resource has a mean of 4 feature-value pairs. The time measurements has been performed by evaluating time between the execution of the considered tool and its termination, while the memory measurement has been done by polling the used memory and catching the greatest value during the execution of each tool.

## 4.2 Results

For quality assessment, we compare the results of HMatch4 and HMatch3 against the ground truth produced by the human users of our crowdsourcing system. The decision to exclude LogMap and SLINT+ arises from the fact that they are proposed to find different representations of the same resource and not to perform similarity recognition, which means that i) the produced values are a representation of *identity* or candidate identity and ii) each resource appears in the results only compared to its best match. In order to make the results produced by HMatch4 and HMatch3 comparable to the ground truth, we converted the continuous values of our tools to the four similarity classes produced by the crowdsourcing system. This has been done creating 4 intervals and 4 association rules mapping each interval to the corresponding similarity class:  $[0, 0.17) \rightarrow U$ ,  $[0.17, 0.5) \rightarrow QS$ ,  $[0.5, 0.8) \rightarrow VS$ ,  $[0.8, 1] \rightarrow E$ .

The comparison between HMatch4 and HMatch3 is based on 3 measures: *precision*, *recall* and *F-measure* (Figure 4). Given the set of mappings in a specific class produced by the automatic tool as  $T$  and the set of mappings of the ground truth in the same class as  $G$ , the precision value is computed as  $\frac{T \cap G}{T}$  and recall value is computed as  $\frac{T \cap G}{G}$ , while F-measure is the harmonic mean between precision and recall. The obtained results show that HMatch4 and HMatch3 are very similar in both precision and recall values. We note that both tools behave exactly as humans on inequality recognition, while they have different perceptions on the other similarity classes. This is probably due to human evaluation of similarity that tends to discriminate the importance of similarity based on the name of the feature. We also considered a different comparison approach by converting the classes of the crowdsourcing system to values in the interval  $[0, 1]$ , in order to perform an overall accuracy measure. This measure has been computed as the inverse of the mean distance of the results produced by our tools against the ground truth. The results of the crowdsourcing system have been converted as follows:  $E \rightarrow 1$ ,  $VS \rightarrow 0.66$ ,  $QS \rightarrow 0.33$ ,  $U \rightarrow 0$ . The accuracy values obtained are 0.886 for HMatch3 and 0.890 for HMatch4. Also in this case the results are almost the same.

Finally, we present the results of the scalability evaluation. All the tests have been executed on a 4-core Intel(R) Xeon(R) processor (model E5-1620) with a frequency of 3.60GHz and 16 GB of total RAM memory. The results shown in Figure 5 (execution time) represent the trend of the 4 considered tools for the time consumption: the y-axis is the log-scale of the required time while the x-axis is the size of the dataset represented by the maximum number of possible comparisons (i.e., for a dataset of  $N$  resources the x-axis shows  $N \cdot N$ ).

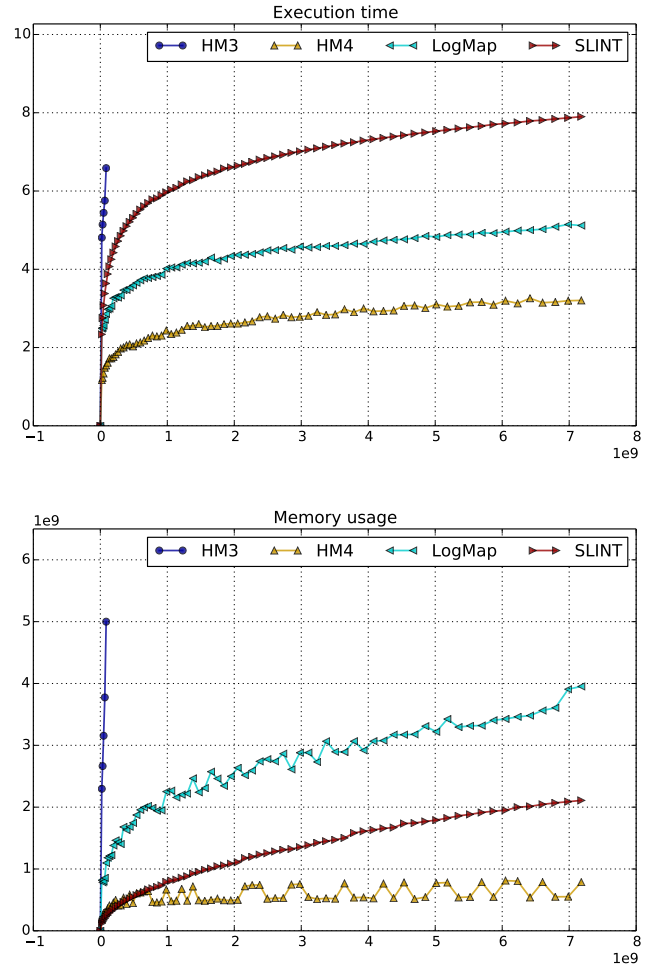


Figure 5: Execution time and memory usage for HMatch3, HMatch4, LogMap, and SLINT+

In Figure 5 (memory usage) the memory consumption of the 4 considered tools is shown. In this case, the y-axis shows the occupied memory in bytes while the x-axis represents the size of the dataset (intended again as the greatest number of possible comparisons). We note that the HMatch3 tool is executed just until the 161<sup>th</sup> replication due to the excessive required time and memory. In all the executed test cases HMatch4 performed as the best tool between the considered counterparts. In the largest test case, we matched  $N \cdot N$  resources with  $N = 85144$  (401775 features, 888293 triples): SLINT+ required 2697 seconds, LogMap required 166 seconds and HMatch4 completed the comparison in 24 seconds.

## 5. RELATED WORK

In the recent years, a lot of research effort has been focused on data matching with a specific attention to instance matching in the framework of the Semantic Web. Most of the existing solutions have been conceived to deal with the so-called *identity-recognition* problem, where the target is to detect when different descriptions extracted from inde-

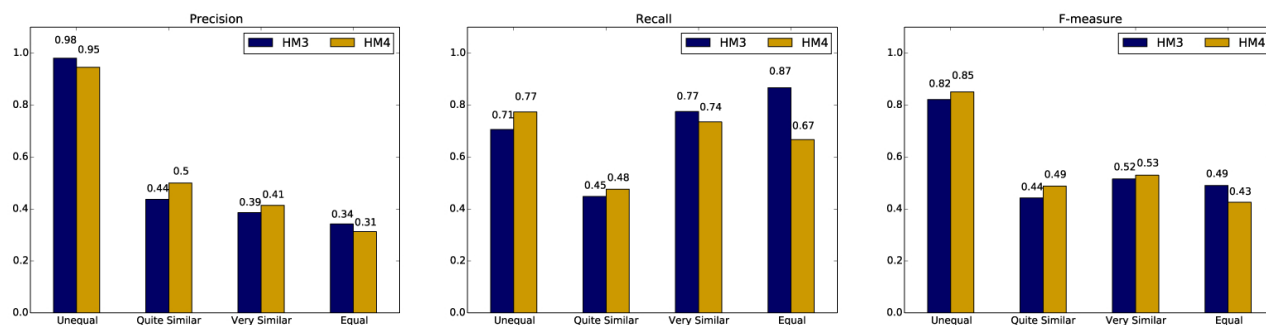


Figure 4: Precision, Recall and F-measure of HMatch3 and HMatch4

pendent web repositories refer to the same individual. On this research line, a reference survey of existing techniques and tools can be found in [5]. The creation of an instance-matching track within the context of the well-known OAEI initiative<sup>1</sup> is a further message that emphasizes the growing attention about the data matching issues. Examples of interesting tools that have been emerging from the OAEI competition are LogMap [7] and SLINT+ [8]. Yet, the focus of these proposed tools is the capability to recognize identities (i.e., *same-as* links) between pairs of web objects.

Approaches based on feature similarity are also relevant with respect to our HMatch4. In this kind of solutions, the objects to match are described through (numeric) feature vectors and the similarity degree is calculated in terms of distances in a  $n$ -dimensional space by relying on vector calculus operations. Vector-based matching techniques are usually employed in image similarity recognition and in nearest neighbor search where the items to compare are characterized by feature vectors with  $n$  numeric coordinates and the mapping within a  $n$ -dimensional space is straightforward [9]. For application of vector-based techniques to web-of-data matching, a transformation of (string-based) feature-value pairs into (numeric) feature vectors is required, but such a kind of transformation is not straightforward.

Finally, the possibility to use hashing solutions to index the object features to match is not completely new in the literature about data matching [6]. The capability to create an efficient data structure for storing similar values in neighbor positions of the index is promising and solutions in this direction are currently appearing [1]. We plan to integrate the use of hashing data structures into HMatch4. We will investigate this issue in the next-future research activities to further increase the performance of execution and assessment steps of the HMatch4 process (see Section 6).

**Original contribution.** With respect to all the above solutions, the peculiarity of HMatch4 is on the goal of the matching process rather than on the novelty of the proposed techniques. In HMatch4, the target is similarity recognition, based on evaluating the relevance of shared subset of feature-values in the different wri specifications. The use of an index structure is adopted to avoid a direct pair-by-pair comparison of all the items to match, with the aim at improving the overall performance of the matching process.

<sup>1</sup><http://oaei.ontologymatching.org/>.

## 6. CONCLUDING REMARKS

In this paper, we presented HMatch4 and related techniques for similarity recognition. HMatch4 has been conceived for application in those contexts where the goal is to evaluate the similarity degree among description of different individuals like for example dimension-based data classification and web data summarization [4]. In future work, we plan to investigate the extension of HMatch4 to support approximate matching. The idea is to enforce hashing techniques that preserve “near” index positions when “near” feature values are recognized. The investigation of hashing techniques based on tree structures is also in the research agenda of HMatch4 for efficient indexing of feature-value pairs.

## 7. REFERENCES

- [1] A. Andoni and P. Indyk. Near-optimal Hashing Algorithms for Approximate nearest Neighbor in high Dimensions. In *Proc. of the 47th IEEE FOCS*, 2006.
- [2] D. Bianchini, C. Cappiello, V. De Antonellis, and B. Pernici. P2S: A Methodology to Enable Inter-organizational Process Design through Web Services. In *Proc. of the 21st Int. CAiSE*, 2009.
- [3] S. Castano, A. Ferrara, S. Montanelli, and G. Varese. Ontology and Instance Matching. In *Knowledge-driven multimedia information extraction and ontology evolution*. Springer, 2011.
- [4] A. Ferrara, L. Genta, and S. Montanelli. Linked Data Classification: a Feature-based Approach. In *Proc. of the 3rd EDBT LWDW Workshop*, 2013.
- [5] Ferrara, A. and Nikolov, A. and Scharffe, F. Data Linking for the Semantic Web. *Int. Journal on Semantic Web and Information Systems*, 7(3), 2011.
- [6] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proc. of the 25th VLDB Conference*, 1999.
- [7] E. Jiménez-Ruiz, B. C. Grau, Y. Zhou, and I. Horrocks. Large-scale Interactive Ontology Matching: Algorithms and Implementation. In *Proc. of the 20th ECAI*, 2012.
- [8] K. Nguyen, R. Ichise, and B. Le. SLINT: A Schema-Independent Linked Data Interlinking System. In *Proc. of the 7th Int. Workshop on Ontology Matching*, 2012.
- [9] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and Efficiency in high Dimensional nearest Neighbor Search. In *Proc. of ACM SIGMOD*, 2009.

# Mining of Diverse Social Entities from Linked Data

Alfredo Cuzzocrea  
ICAR-CNR & Univ. of Calabria  
Rende (CS), Italy  
cuzzocrea@si.deis.unical.it

Carson K. Leung  
University of Manitoba  
Winnipeg, MB, Canada  
kleung@cs.umanitoba.ca

Syed K. Tanbeer  
University of Manitoba  
Winnipeg, MB, Canada  
tanbeer@cs.umanitoba.ca

## ABSTRACT

Nowadays, high volumes of valuable data can be easily generated or collected from various data sources at high velocity. As these data are often related or linked, they form a web of linked data. Examples include semantic web and social web. The social web captures social relationships that link people (i.e., social entities) through the World Wide Web. Due to the popularity of social networking sites, more people have joined and more online social interactions have taken place. With a huge number of social entities (e.g., users or friends in social networks), it becomes important to analyze high volumes of linked data and discover those diverse social entities. In this paper, we present (i) a tree-based mining algorithm called **DF-growth**, along with (ii) its related data structure called **DF-tree**, which allow users to effectively and efficiently mine diverse friends from social networks. Results of our experimental evaluation showed both the time- and space-efficiency of our scalable DF-growth algorithm, which makes good use of the DF-tree structure.

## Categories and Subject Descriptors

E.2 [Data]: Data Storage Representations—*linked representations*; H.2.8 [Database Management]: Database Applications—*data mining*; J.2 [Computer Applications]: Social and Behavioral Sciences

## General Terms

Algorithms; Design; Experimentation; Management; Performance; Theory

## Keywords

Data mining, friendship patterns, diverse friends, linked data, social networks, extending database technology, database theory

## 1. INTRODUCTION & RELATED WORK

Nowadays, with the advances in technology, high volumes of valuable data—such as blogs, forums, wikis, and users’ reviews—can be easily generated or collected from various data sources. These data are often related or linked, and thus form a web of linked data [3]. Over the few years, researchers have modelled, queried, and reasoned these linked web data. For instance, Pernelle and Saïs [15] focused on classification rule learning for linked data. Ferrara et al. [9] proposed a feature-based approach to classify linked data.

A social web is an instance of a web of linked data. Such a social web can be viewed as a collection of social relationships that link social entities (e.g., users). In recent years, researchers have exploited the social perspective or social phenomena [4, 8] in this web of linked data (e.g., for the relevant problem of detecting communities over social and information networks [6, 7, 20]). Intuitively, social networks [14] are made of social entities who are linked by some specific types of relationships (e.g., friendship, common interest, kinship). Facebook, Google+, LinkedIn, Twitter and Weibo [17, 22] are some examples of social networks. Within these networks, a user  $f_i$  usually can create a personal profile, add other users as friends, endorse their skills/expertise, exchange messages among friends. These social networks may consist of thousands or millions of users; each user  $f_i$  can have different number of friends. Among them, some are more important (or influential, prominent, and/or active in a wide range of domains) than others [2, 12, 13, 19, 23]. Recognizing these diverse friends can provide valuable information for various real-life applications when analyzing and mining high volumes of valuable social network data.

Over the past few years, several data mining techniques [11, 16, 21] have been developed to help users extract implicit, previously unknown, and potentially useful information about the important friends. Recent works on social network mining include the discovery of strong friends [5] and significant friends [18] based on the degree of one-to-one interactions (e.g., based on the number of postings to a friend’s wall).

However, in some situations, it is also important to discover users who (i) are influential in the social networks, (ii) have high level of expertise in some domains, and/or (iii) have diverse interest in multiple domains. In other words, users may want to find important friends based on their influence, prominence, and/or diversity. For instance, some users may be narrowly interested in one specific domain (e.g., computers). Other users may be interested in a wide range of domains (e.g., computers, music, sports),



but their expertise level may vary from one domain to another (e.g., a user  $f_i$  may be a computer expert but only a beginner in music).

In this paper, we propose a tree-based mining algorithm to find from social networks for those diverse friends, who are highly influential across multiple social network domains. To this end, one of our *key contributions* is an efficient tree-based algorithm called **DF-growth** for mining diverse friends from social networks. DF-growth takes into account multiple properties (e.g., influence, prominence, and/or diversity) of friends in the networks. Another *key contribution* is a prefix-tree based structure called **DF-tree** for capturing the social network data in a memory-efficient manner. Once the DF-tree is constructed, DF-growth computes the diversity of users based on both their influence and prominence to mine diverse groups of friends.

The remainder of this paper is organized as follows. The next section introduces the notion of diverse friends. Section 3 presents our DF-growth algorithm, which mines diverse friends from our DF-tree. Experimental results are reported in Section 4. Conclusions and future work are given in Section 5.

## 2. DIVERSE FRIENDS IN SOCIAL NETWORKS

This section presents the concept of diverse friends in social networks. Let us consider a social network on three different domains (domains  $D_1, D_2, D_3$ ) and seven individuals—Antonios, Barbara, Georgios, Dimitrios, Evgenios, Zoe, and Hebe—with prominence values in each domain, as shown in Table 1. Each *domain* represents a sub-category (e.g., sports, arts, education) of interest. The **prominence value** of an individual reveals his level of expertise (e.g., importance, weight, value, reputation, belief, position, status, or significance) in a domain. In other words, the prominence value indicates how important, valued, significant, or well-positioned the individual is in each domain. The prominence value can be measured by using a common scale, which could be (i) specified by users or (ii) automatically calculated based on some user-centric parameters (e.g., connectivity, centrality, expertise in the domain, years of membership in the domain, degree of involvement in activities in the domain, numbers of involved activities in the domain). In this paper, the prominence value is normalized into the range (0, 1]. As the same individual may have different levels of expertise in different domains, his corresponding prominence value may vary from one domain to another. For instance, prominence value  $Prom_{D_1}$ (Antonios) of Antonios in domain  $D_1$  is 0.45, which is different from  $Prom_{D_2}$ (Antonios) = 0.60. Moreover,  $Prom_{D_1}$ (Antonios) is higher than  $Prom_{D_1}$ (Georgios) = 0.20, implying that Antonios is more influential than Georgios in domain  $D_1$ .

Similar to the existing settings of a social network [5, 11, 18], let  $F = \{f_1, f_2, \dots, f_m\}$  be a set of social entities/friends in a social network. An *interest-group list*  $L \subseteq F$  is a list of individuals who are connected as friends due to some common interests. Let  $G = \{f_1, f_2, \dots, f_k\} \subseteq F$  be a **group of friends** (i.e., friend group) with  $k$  friends. Then,  $Size(G) = k$ , which represents the number of individuals in  $G$ . A *friend network*  $F_{SN} = \{L_1, L_2, \dots, L_n\}$  is the set of all  $n$  interest-group lists in the entire social network. These lists belong to some domains, and each domain

**Table 1: Prominence of friends**

Friend ( $f_i$ )	Prominence $Prom(f_i)$		
	Domain $D_1$	Domain $D_2$	Domain $D_3$
Antonios	0.45	0.60	0.50
Barbara	0.90	0.70	0.30
Georgios	0.20	0.60	0.70
Dimitrios	0.30	0.50	0.40
Evgenios	0.50	0.40	0.45
Zoe	0.42	0.24	0.70
Hebe	0.57	0.10	0.20

**Table 2: Lists of interest groups in  $F_{SN}$**

Domain	Interest-group list $L_j$
$D_1$	$L_1 = \{\text{Antonios, Barbara}\}$
	$L_2 = \{\text{Antonios, Barbara, Dimitrios}\}$
	$L_3 = \{\text{Georgios, Dimitrios}\}$
$D_2$	$L_4 = \{\text{Barbara, Georgios, Dimitrios}\}$
	$L_5 = \{\text{Barbara, Georgios, Evgenios}\}$
	$L_6 = \{\text{Barbara, Hebe}\}$
	$L_7 = \{\text{Georgios, Evgenios}\}$
$D_3$	$L_8 = \{\text{Antonios, Georgios}\}$
	$L_9 = \{\text{Antonios, Georgios, Evgenios}\}$
	$L_{10} = \{\text{Antonios, Zoe}\}$

contains at least one list. The set of lists in a particular domain  $D$  is called a *domain database* (denoted as  $F_D$ ). Here, we assume that there exists an interest-group list in every domain. The *projected list*  $F_D^G$  of  $G$  in  $F_D$  is the set of lists in  $F_D$  that contains group  $G$ . The frequency  $Freq_D(G)$  of  $G$  in  $F_D$  indicates the number of lists  $L_j$ 's in  $F_D^G$ , and the frequencies of  $G$  in multiple domains are represented as  $Freq_{D_1, 2, \dots, d}(G) = \langle Freq_{D_1}(G), Freq_{D_2}(G), \dots, Freq_{D_d}(G) \rangle$ .

EXAMPLE 2.1. Consider  $F_{SN}$  shown in Table 2, which consists of  $n=10$  interest-group lists  $L_1, \dots, L_{10}$  for  $m=7$  social individuals/friends in Table 1. Each row in the table represents the list of an interest group. These 10 interest groups are distributed into  $d=3$  domains  $D_1, D_2$  and  $D_3$ . For instance,  $F_{D_1} = \{L_1, L_2, L_3\}$ . For group  $G = \{\text{Georgios, Evgenios}\}$ , its  $Size(G)=2$ . As its projected lists on the 3 domains are  $F_{D_1}^G = \emptyset$ ,  $F_{D_2}^G = \{L_6, L_7\}$  and  $F_{D_3}^G = \{L_8\}$ , its frequencies  $Freq_{D_1, 2, 3}(G) = \langle 0, 2, 1 \rangle$ .  $\square$

DEFINITION 2.1. The **prominence value**  $Prom_D(G)$  of a friend group  $G$  in a single domain  $D$  is defined as the average of all prominence values for all the friends in  $G$ :

$$Prom_D(G) = \frac{\sum_{i=1}^{Size(G)} Prom_D(f_i)}{Size(G)}. \quad (1)$$

Then, prominence values  $Prom_{D_1, 2, \dots, d}(G)$  of a friend group  $G$  in multiple domains are represented as  $Prom_{D_1, 2, \dots, d}(G) = \langle Prom_{D_1}(G), Prom_{D_2}(G), \dots, Prom_{D_d}(G) \rangle$ .  $\square$

EXAMPLE 2.2. Revisit  $F_{SN}$  in Table 2. The prominence value of friend group  $G = \{\text{Georgios, Evgenios}\}$  in  $D_1 = \frac{Prom_{D_1}(\text{Georgios}) + Prom_{D_1}(\text{Evgenios})}{Size(G)} = \frac{0.20 + 0.50}{2} = 0.35$ . We apply similar computation on the other two domains  $D_2$  and  $D_3$  to get  $Prom_{D_1, 2, 3}(G) = \langle 0.35, \frac{0.60 + 0.40}{2}, \frac{0.70 + 0.45}{2} \rangle = \langle 0.35, 0.5, 0.575 \rangle$ .  $\square$

DEFINITION 2.2. The **influence**  $\text{Inf}_D(G)$  of a group  $G$  of social entities/friends in a domain  $D$  in  $F_D$  is defined as the product of the prominence value of  $G$  in the domain  $D$  and its frequency in the domain database  $F_D$ , i.e.,

$$\text{Inf}_D(G) = \text{Prom}_D(G) \times \text{Freq}_D(G). \quad (2)$$

The influence  $\text{Inf}_{D_1,2,\dots,d}(G)$  of  $G$  in multiple domains is then represented as  $\text{Inf}_{D_1,2,\dots,d}(G) = \langle \text{Inf}_{D_1}(G), \text{Inf}_{D_2}(G), \dots, \text{Inf}_{D_d}(G) \rangle$ .  $\square$

EXAMPLE 2.3. Continue with Example 2.2. Recall that  $\text{Prom}_{D_1,2,3}(G) = \langle 0.35, 0.5, 0.575 \rangle$ . Recall from Example 2.1 that  $\text{Freq}_{D_1,2,3}(G) = \langle 0, 2, 1 \rangle$ . Then, the overall influence of  $G$  in all 3 domains can be calculated as  $\text{Inf}_{D_1,2,3}(G) = \langle 0.35 \times 0, 0.5 \times 2, 0.575 \times 1 \rangle = \langle 0, 1, 0.575 \rangle$ .  $\square$

DEFINITION 2.3. The **diversity**  $\text{Div}(G)$  of a group  $G$  of social entities/friends among all  $d$  domains in  $F_{SN}$  is defined as the average of all the influence values of  $G$  in all domains in the social network:

$$\text{Div}(G) = \frac{\sum_{j=1}^d \text{Inf}_{D_j}(G)}{d}. \quad (3)$$

EXAMPLE 2.4. Continue with Example 2.3. Recall that  $\text{Inf}_{D_1,2,3}(G) = \langle 0, 1, 0.575 \rangle$ . Then, the diversity of  $G$  in these  $d=3$  domains in  $F_{SN}$  is  $\text{Div}(G) = \frac{0+1+0.575}{3} = 0.525$ .  $\square$

Here, a group  $G$  of friends in a social network  $F_{SN}$  is considered **diverse** if its diversity value  $\text{Div}(G) \geq$  user-specified minimum threshold  $\text{minDiv}$ , which can be expressed as an absolute (non-negative real) number or a relative percentage (with respect to the size of  $F_{SN}$ ). Given  $F_{SN}$  and  $\text{minDiv}$ , the research problem of **mining diverse friends from social networks** is to find every group  $G$  of friends having  $\text{Div}(G) \geq \text{minDiv}$ .

EXAMPLE 2.5. Let group  $G = \{\text{Georgios}, \text{Evgenios}\}$ . Recall from Example 2.4 that diversity  $\text{Div}(G) = 0.525$ . Given (i)  $F_{SN}$  in Table 2 and (ii) the user-specified  $\text{minDiv}=0.5$ ,  $G$  is *diverse* because  $\text{Div}(G)=0.525 \geq 0.5=\text{minDiv}$ .

However, group  $G' = \{\text{Evgenios}\}$ , such that  $G' \subseteq G$  is *not* diverse because  $\text{Div}(G') = \frac{(0.5 \times 0) + (0.4 \times 2) + (0.45 \times 1)}{3} = \frac{0+0.8+0.45}{3} = 0.417 < \text{minDiv}$ .  $\square$

Note that, when mining frequent patterns, the frequency/support measure [1, 10] satisfies the downward closure property (i.e., all supersets of an infrequent patterns are infrequent). This helps reduce the search/solution space by pruning infrequent patterns, which in turn speeds up the mining process.

However, mining diverse friends is different from mining frequent patterns. As observed from Example 2.5 that group  $G' = \{\text{Evgenios}\}$  is not diverse but its super-group  $G = \{\text{Georgios}, \text{Evgenios}\}$  is diverse. In other words, diversity does *not* satisfy the downward closure property (i.e., if a group is not diverse, then *not* all of its super-groups are guaranteed to be diverse).

### 3. MINING DIVERSE FRIENDS

Given that diversity does *not* satisfy the downward closure property, we cannot prune those groups that are not diverse. Hence, the mining of diverse friends can be challenging. To handle this challenge, for each domain  $D$ , we identify

the **(global) maximum prominence value**  $\text{GMProm}_D$  among all friends. Then, for each friend  $f_i$ , we calculate an upper bound of the influence value  $\text{Inf}_D^U(f_i)$  by multiplying  $\text{GMProm}_D$  (instead of the actual  $\text{Prom}_D(f_i)$ ) with the corresponding frequency  $\text{Freq}_D(f_i)$ . The upper bound of diversity value  $\text{Div}^U(f_i)$  can then be computed by using  $\text{Inf}_D^U(f_i)$ .

LEMMA 3.1. Let  $G$  be a group of friends in  $F_{SN}$  such that a friend  $f_i \in G$ . If  $\text{Div}^U(f_i) < \text{minDiv}$ , then  $\text{Div}(G)$  must also be less than  $\text{minDiv}$ .  $\square$

EXAMPLE 3.1. Let us revisit  $F_{SN}$  in Table 2. Global maximum prominence values are  $\text{GMProm}_{D_1}=0.90$ ,  $\text{GMProm}_{D_2}=0.70$ , and  $\text{GMProm}_{D_3}=0.70$ . Recall from Example 2.5 that  $\text{Freq}_{D_1,2,3}(\{\text{Evgenios}\}) = \langle 0, 2, 1 \rangle$ . Then, we can compute  $\text{Div}^U(\{\text{Evgenios}\}) = \frac{(0.90 \times 0) + (0.70 \times 2) + (0.70 \times 1)}{3} = 0.7 \geq \text{minDiv}$ . So, we do not prune  $\{\text{Evgenios}\}$  to avoid missing its super-group  $\{\text{Georgios}, \text{Evgenios}\}$ , which is diverse. Similarly,  $\text{Div}^U(\{\text{Zoe}\}) = \frac{(0.90 \times 0) + (0.70 \times 0) + (0.70 \times 1)}{3} = 0.23 < \text{minDiv}$ . Due to Lemma 3.1, we prune Zoe as none of its super-groups can be diverse.  $\square$

### 3.1 Construction of a DF-tree Structure

Our proposed DF-growth algorithm takes (i) a friend network  $F_{SN}$  and (ii) a user-specified  $\text{minDiv}$  threshold as two input parameters to construct a DF-tree as follows. It first scans  $F_{SN}$  to calculate  $\text{Freq}_{D_j}(f_i)$  for each friend  $f_i$  in each domain  $D_j$ . For each  $f_i$ , DF-growth then uses  $\text{GMProm}_D$  to compute the upper bound of the diversity value  $\text{Div}^U(f_i)$ , which is used to prune groups of friends who are not potentially diverse. Every potentially diverse friend  $f_i$ , along with its  $\text{Freq}_{D_1,\dots,d}(f_i)$ , is stored in the header table.

Then, DF-growth scans  $F_{SN}$  the second time to capture the important information about potentially diverse friends in a user-defined order in the DF-tree. Each tree node consists of (i) a friend name and (ii) its frequency counters for all  $d$  domains in the respective path. The basic construction process of a DF-tree is similar to that of the FP-tree [10]. A key difference is that, rather than using only a single frequency counter capturing either the maximum or average frequency for all domains (which may lead to loss of information), we use  $d$  frequency counters capturing the frequency for all  $d$  domains. See Example 3.2.

EXAMPLE 3.2. To construct a DF-tree for  $F_{SN}$  shown in Table 2 when  $\text{minDiv}=0.5$ , DF-growth scans  $F_{SN}$  to compute (i)  $\text{GMProm}_{D_1,2,3} = \langle 0.9, 0.7, 0.7 \rangle$  for all  $d=3$  domains, (ii) frequencies of each of the 7 friends in  $d=3$  domains (e.g.,  $\text{Freq}_{D_1,2,3}(\{\text{Antonios}\}) = \langle 2, 0, 3 \rangle$ ), (iii) upper bound of diversity values of all 7 friends (e.g.,  $\text{Div}^U(\{\text{Antonios}\}) = \frac{(0.9 \times 2) + (0.7 \times 0) + (0.7 \times 3)}{3} = 1.3$  using  $\text{Inf}_{D_1,2,3}^U(\{\text{Antonios}\})$ ). Based on Lemma 3.1, we safely remove Zoe and Hebe having  $\text{Div}^U(\{\text{Zoe}\})=0.23$  and  $\text{Div}^U(\{\text{Hebe}\})=0.23$  both below  $\text{minDiv}$  as their super-groups cannot be diverse. So, the header table includes only the remaining 5 friends—sorted in some order (e.g., lexicographical order of friend names)—with their  $\text{Freq}_{D_1,2,3}(\{f_i\})$ . To facilitate a fast tree traversal, like the FP-tree, the DF-tree also maintains horizontal node traversal pointers from the header table to nodes of the same  $f_i$ .

Our DF-growth algorithm then scans each  $L_j \in F_{SN}$ , removes any friend  $f_i \in L_j$  having  $\text{Div}^U(f_i) < \text{minDiv}$ , sorts

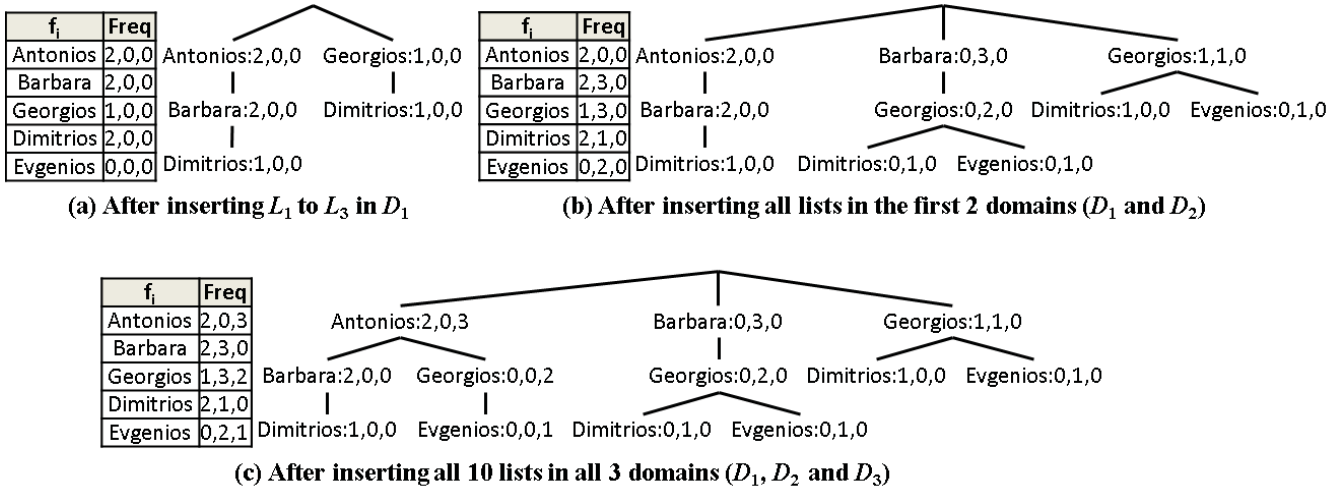


Figure 1: Construction of a DF-tree.

the remaining friends according to the order in the header table, and inserts the sorted list into the DF-tree. Each tree node captures (i)  $f_i$  representing the group  $G$  consisting of all friends from the root to  $f_i$  and (ii) its frequencies in each domain  $Freq_{D_{1,2,3}}(G)$ . For example, the rightmost node Evgenios:0,1,0 of the DF-tree in Figure 1(b) captures  $G=\{\text{Georgios, Evgenios}\}$  and  $Freq_{D_{1,2,3}}(G)=(0, 1, 0)$ . Tree paths of common prefix (i.e., same friends) are shared, and their corresponding frequencies are added. See Figures 1(a), 1(b), and 1(c) for DF-trees after reading all interest-group lists in domain  $D_1$ , both  $D_1$  and  $D_2$ , as well as the entire  $F_{SN}$ , respectively.  $\square$

With this tree construction process, the size of the DF-tree for  $F_{SN}$  with a given  $minDiv$  is observed to be bounded above by  $\sum_{L_j \in F_{SN}} |L_j|$ .

### 3.2 Mining of All Diverse Friend Groups

After constructing the DF-tree, our DF-growth algorithm recursively mines/discovers diverse friend groups by building projected and conditional trees in a fashion similar to that of FP-growth [10].

Recall that  $Div(G)$  computed based on  $Prom_D(G)$  does not satisfy the downward closure property. To facilitate pruning, we use  $GMProm_D(f_i)$  to compute  $Div^U(f_i)$ , which then satisfies the downward closure property. However, if  $Div^U(G)$  was computed as an upper bound to super-group  $G$  of  $f_i$ , then it may overestimate diversity of  $G$  and may lead to false positives. To reduce the number of false positives, DF-growth uses the **local maximum prominence value**  $LMProm_D(G) = \max_{f_i \in F_D^G} \{Prom_D(G)\}$  for the projected and conditional trees for  $G$ . See Lemma 3.2 and Example 3.3.

**LEMMA 3.2.** *The diversity value of a friend group  $G$  computed based on  $LMProm_D(G)$  is a tighter upper bound than  $Div^U(G)$  computed based on  $GMProm_D$ .  $\square$*

**EXAMPLE 3.3.** Let us continue Example 3.2. To mine potentially diverse friend groups from the DF-tree in Figure 1(b) using  $minDiv = 0.5$ , DF-growth first builds the

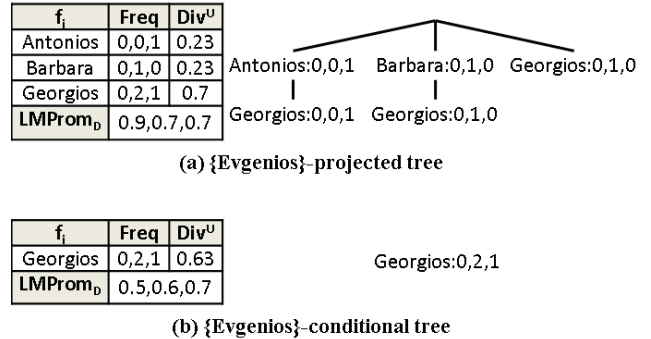


Figure 2: Tree-based mining of diverse friend groups.

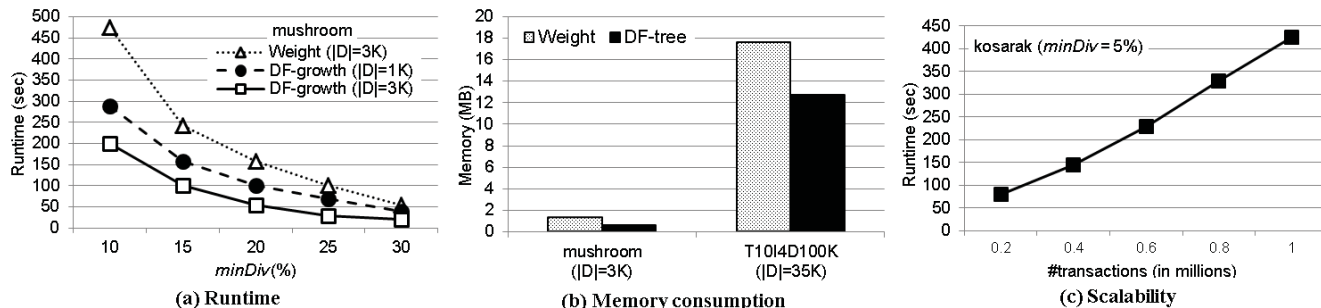
{Evgenios}-projected tree—as shown in Figure 2(a)—by extracting the paths  $\langle \text{Antonios, Georgios, Evgenios} \rangle:0,0,1$ ,  $\langle \text{Barbara, Georgios, Evgenios} \rangle:0,1,0$  and  $\langle \text{Georgios, Evgenios} \rangle:0,1,0$  from the DF-tree in Figure 1(b). For  $F_{D_{1,2,3}}^{\text{Evgenios}} = \{\text{Antonios, Barbara, Georgios, Evgenios}\}$ , our DF-growth algorithm uses  $LMProm_{D_{1,2,3}}(F_{D_{1,2,3}}^{\text{Evgenios}}) = \langle 0.9, 0.7, 0.7 \rangle$  to compute the tightened  $Div^U(G)$  such that the tightened  $Div^U(\{\text{Antonios, Evgenios}\}) = \frac{(0.9 \times 0) + (0.7 \times 0) + (0.7 \times 1)}{3} = 0.23 < minsup$ .

As  $Div^U(\{\text{Antonios, Evgenios}\})$  and  $Div^U(\{\text{Barbara, Evgenios}\})$  are both below  $minsup$ , DF-growth prunes Antonios and Barbara from the {Evgenios}-projected tree to get the {Evgenios}-conditional tree as shown in Figure 2(b). Due to pruning, our DF-growth algorithm recomputes (i) the local maximum prominence value  $LMProm_{D_{1,2,3}}(F_{D_{1,2,3}}^{\text{Evgenios}}) = \langle 0.5, 0.6, 0.7 \rangle$  and (ii) the tightened  $Div^U(\{\text{Georgios, Evgenios}\}) = \frac{(0.5 \times 0) + (0.6 \times 2) + (0.7 \times 1)}{3} = 0.63$  for the updated  $F_{D_{1,2,3}}^{\text{Evgenios}} = \{\text{Georgios, Evgenios}\}$ . This completes the mining for {Evgenios}.

Next, DF-growth builds {Dimitrios}-, {Georgios}- and {Barbara}-projected trees as well as their conditional trees, from which potentially diverse friend groups can be mined. Finally, our DF-growth algorithm computes the true diversity value  $Div(G)$  for each of these mined groups to check if it is truly diverse (i.e., to remove all false positives).  $\square$

**Table 3: Dataset characteristics**

Dataset	$n$ =#transactions	$m$ =#domain items	Max trans. length	Avg trans. length	Density
mushroom	8,124	119	23	23.0	Dense
T1014D100K	100,000	870	29	10.1	Sparse
kosarak	990,002	41,270	2498	8.1	Sparse


**Figure 3: Experimental results.**

### 3.3 Removal of Non-diverse Friend Groups

Our DF-growth algorithm makes good use of the global and local maximum prominence values of friend groups as upper bounds to diversity values of friend groups. Consequently, the algorithm discovers all truly diverse friend groups (i.e., *no* false negatives). However, it also discover some “potentially diverse” friend groups that are not truly diverse (i.e., some false positives). Hence, as its final step, our algorithm computes the true diversity values  $Div(G)$  for each of these mined groups to check if it is truly diverse (i.e., to remove all false positives).

EXAMPLE 3.4. Let us continue Example 3.3. After mining potentially diverse friend groups from {Evgenios}-, {Dimitrios}-, {Georgios}- and {Barbara}-projected trees as well as their conditional trees, our DF-growth algorithm computes the true diversity value  $Div(G)$  for each of the mined groups to check if it is truly diverse (i.e., to remove all false positives). □

## 4. EXPERIMENTAL EVALUATION

To evaluate the effectiveness of our proposed DF-growth algorithm and its associated DF-tree structure, we compared them with a closely related *weighted* frequent pattern mining algorithm called Weight [24] (but it does not use different weights for individual items). As Weight was designed for frequent pattern mining (instead of social network mining), we apply those datasets commonly used in frequent pattern mining for a fair comparison: (i) IBM synthetic datasets (e.g., T1014D100K) and (ii) real datasets (e.g., mushroom, kosarak) from the Frequent Itemset Mining Dataset Repository (<http://fimi.ua.ac.be/data>). See Table 3 for more detail. Items in transactions in these datasets are mapped into friends in interest-group lists. To reflect the concept of *domains*, we subdivided the datasets into several batches. Moreover, a random number in the range  $(0, 1]$  is generated as a prominence value for each friend in every domain.

All programs were written in C++ and run on the Windows XP operating system with a 2.13 GHz CPU and 1 GB

main memory. The runtime specified indicates the total execution time (i.e., CPU and I/Os). The reported results are based on the average of multiple runs for each case. We obtained consistent results for all of these datasets.

### 4.1 Runtime

First, we compared the runtime of DF-growth (which includes the construction of the DF-tree, the mining of potentially diverse friend groups from the DF-tree, and the removal of false positives) with that of Weight. Figure 3(a) shows the results for a dense dataset (mushroom), which were consistent with those for sparse datasets (e.g., T1014D100K). Due to page limitation, we omit the results for sparse datasets. But, runtimes of both algorithms increased when mining larger datasets (social networks), more batches (domains), and/or with lower *minDiv* thresholds. Between the two algorithms, our tree-based DF-growth algorithm outperformed the Apriori-based Weight algorithm. Note that, although FP-growth [10] is also a tree-based algorithm, it was *not* design to capture weights. To avoid distraction, we omit experimental results on FP-growth and only show those on Weight (which captures weights).

### 4.2 Memory Consumption

Second, we evaluated the memory consumption. Figure 3(b) shows the amount of memory required by our DF-tree for capturing the content of social networks with the lowest *minDiv* threshold (i.e., without removing any friends who were not diverse). Although this simulated the worst-case scenario for our DF-tree, DF-tree was observed (i) to consume a reasonable amount of memory and (ii) to require less memory than Weight (because our DF-tree is compact due to the prefix sharing).

### 4.3 Scalability

Third, we tested the scalability of our DF-growth algorithm by varying the number of transactions (interest-group lists). We used the kosarak dataset as it is a huge sparse dataset with a large number of distinct items (individual users). We divided this dataset into five portions, and each por-



tion is subdivided into multiple batches (domains). We set  $minDiv=5\%$  of each portion. Figure 3(c) shows that, when the size of the dataset increased, the runtime also increased proportionally implying that DF-growth is scalable.

#### 4.4 Summary on Evaluation Results

Experimental results on (i) runtime, (ii) memory consumption (which reveals tree compactness) and (iii) scalability showed that our DF-growth algorithm is time- and space-efficient as well as scalable. As ongoing work, we plan to evaluate the *quality* (e.g., precision) of DF-growth in finding diverse friend groups. Moreover, for a fair comparison with *Weight*, we have used those datasets that are commonly used in frequent pattern mining. As ongoing work, we plan to evaluate DF-growth using real-life social network datasets.

### 5. CONCLUSIONS AND FUTURE WORK

In this paper, we (i) introduced a new notion of *diverse friends* for social networks, (ii) proposed a compact tree structure called *DF-tree* to capture important information from social networks, and (iii) designed a tree-based mining algorithm called *DF-growth* to find diverse (groups of) friends from social networks. Diversity of friends is measured based on their prominence, frequency and influence in different domains on the networks. Although diversity does not satisfy the downward closure property, we managed to address this issue by using the global and local maximum prominence values of users as upper bounds. Experimental results showed that (i) our DF-tree is compact and space-effective and (ii) our DF-growth algorithm is fast and scalable for both sparse and dense datasets. As ongoing work, we conduct more extensive experimental evaluation with various datasets (e.g., real-life social network datasets) and to measure other aspects (e.g., precision) of our DF-growth algorithm in finding diverse friends. We also plan to (i) design a more sophisticated way to measure influence and (ii) incorporate other computational metrics (e.g., popularity, significance, strength) with prominence into our discovery of useful information from social networks.

### 6. ACKNOWLEDGEMENTS

This project is partially supported by NSERC (Canada) and University of Manitoba.

### 7. REFERENCES

- [1] R. Agrawal & R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. VLDB 1994*, pp. 487–499.
- [2] A. Anagnostopoulos, R. Kumar, & M. Mahdian. Influence and correlation in social networks. In *Proc. ACM KDD 2008*, pp. 7–15.
- [3] D. Bianchini, V. de Antonellis, & M. Melchiori. A linked data perspective for collaboration in mashup development. In *Proc. DEXA Workshops 2013*, pp. 128–132.
- [4] D. Bianchini, V. de Antonellis, & M. Melchiori. Exploiting social tagging in web API search. In *Proc. OTM 2013*, pp. 764–771.
- [5] J.J. Cameron, C.K. Leung, & S.K. Tanbeer. Finding strong groups of friends among friends in social networks. In *Proc. SCA 2011*, pp. 824–831.
- [6] A. Cuzzocrea & F. Folino. Community evolution detection in time-evolving information networks. In *Proc. EDBT/ICDT 2013 Workshops (LWDM)*, pp. 93–96.
- [7] A. Cuzzocrea, F. Folino, & C. Pizzuti. *DynamicNet*: an effective and efficient algorithm for supporting community evolution detection in time-evolving information networks. In *Proc. IDEAS 2013*, pp. 148–153.
- [8] R. de Virgilio & A. Maccioni. Generation of reliable randomness via social phenomena. In *Proc. MEDI 2013*, pp. 65–77.
- [9] A. Ferrara, L. Genta, & S. Montanelli. Linked data classification: a feature-based approach. In *Proc. EDBT/ICDT 2013 Workshops (LWDM)*, pp. 75–82.
- [10] J. Han, J. Pei, & Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD 2000*, pp. 1–12.
- [11] F. Jiang, C.K. Leung, & S.K. Tanbeer. Finding popular friends in social networks. In *Proc. SCA 2012*, pp. 501–508.
- [12] K.Y. Kamath, J. Caverlee, Z. Cheng, & D.Z. Sui. Spatial influence vs. community influence: modeling the global spread of social media. In *Proc. ACM CIKM 2012*, pp. 962–971.
- [13] W. Lee, C.K. Leung, J.J. Song, & C.S. Eom. A network-flow based influence propagation model for social networks. In *Proc. SCA 2012*, pp. 601–608.
- [14] C.K. Leung & C.L. Carmichael. Exploring social networks: a frequent pattern visualization approach. In *Proc. IEEE SocialCom 2010*, pp. 419–424.
- [15] N. Pernelle & F. Saïs. Classification rule learning for data linking. In *Proc. EDBT/ICDT 2012 Workshops (LWDM)*, pp. 136–139.
- [16] Y. Ruan, D. Fuhry, & S. Parthasarathy. Efficient community detection in large networks using content and links. In *Proc. ACM WWW 2013*, pp. 1089–1098.
- [17] M. Schaal, J. O’Donovan, & B. Smyth. An analysis of topical proximity in the twitter social graph. In *Proc. SocInfo 2012*, pp. 232–245.
- [18] S.K. Tanbeer, F. Jiang, C.K. Leung, R.K. MacKinnon, & I.J.M. Medina. Finding groups of friends who are significant across multiple domains in social networks. In *Proc. CASoN 2013*, pp. 21–26.
- [19] S.K. Tanbeer, C.K. Leung, & J.J. Cameron. DIFSoN: discovering influential friends from social networks. In *Proc. CASoN 2012*, pp. 120–125.
- [20] Z. Wu, W. Yin, J. Cao, G. Xu, & A. Cuzzocrea. Community detection in multi-relational social networks. In *Proc. WISE 2013*, pp. 43–56.
- [21] T. Yang, P.M. Comar, & L. Xu. Community detection by popularity based models for authored networked data. *Proc. IEEE/ACM ASONAM 2013*, pp. 74–81.
- [22] Q. Yuan, G. Cong, Z. Ma, A. Sun, & N. Magnenat-Thalmann. Who, where, when and what: discover spatio-temporal topics for twitter users. In *Proc. ACM KDD 2013*, pp. 605–613.
- [23] C. Zhang, L. Shou, K. Chen, G. Chen, & Y. Bei. Evaluating geo-social influence in location-based social networks. In *Proc. ACM CIKM 2012*, pp. 1442–1451.
- [24] S. Zhang, C. Zhang, & X. Yan. Post-mining: maintenance of association rules by weighting. *Information Systems*, **28**(7), pp. 691–707 (2003).

# TripleGeo: an ETL Tool for Transforming Geospatial Data into RDF Triples

Kostas Patroumpas<sup>†,§</sup> Michalis Alexakis<sup>§</sup> Giorgos Giannopoulos<sup>§,†</sup> Spiros Athanasiou<sup>§</sup>

<sup>§</sup>Institute for the Management of Information Systems  
"Athena" Research Center, Hellas

<sup>†</sup>School of Electrical and Computer Engineering  
National Technical University of Athens, Hellas

kpatro@dblab.ece.ntua.gr, {alexakis, giann, sathan}@imis.athena-innovation.gr

## ABSTRACT

Integrating data from heterogeneous sources has led to the development of Extract-Transform-Load (ETL) systems and methodologies, as a means of addressing modern interoperability challenges. A few such tools have been available for converting between geospatial formats, but none specifically addressing the emerging needs of geospatially-enabled RDF stores. In this paper, we introduce TripleGeo, an open-source ETL utility that can extract geospatial features from various sources and transform them into triples for subsequent loading into RDF stores. TripleGeo can directly access both geometric representations and thematic attributes either from standard geographic formats or widely used DBMSs. It can also reproject input geometries on-the-fly into a different Coordinate Reference System, before exporting the resulting triples into a variety of notations. Most importantly, TripleGeo supports the recent GeoSPARQL standard endorsed by the Open GeoSpatial Consortium, although it can extract geometries into other vocabularies as well. This tool has been validated against OpenStreetMap layers with millions of geometries, opening up perspectives to add more functionality and to address much bigger data volumes.

## 1. INTRODUCTION

Nowadays, geospatial data is ubiquitous on the Web, either explicitly (through maps or satellite imagery) or implicitly (e.g., via addresses, geotagged photographs, or geolocation hashtags). Spatial information can be found in a variety of data formats, schemas, and heterogeneous platforms, systems, web services, etc. Most of this data still remains in proprietary databases and Geographic Information Systems (GIS) maintained by commercial vendors or governmental agencies. Standardization by the Open GeoSpatial Consortium (OGC) [24] or initiatives for building spatial data infrastructures under the EU INSPIRE Directive [11], pave the way towards geospatial data interoperability and dissemination. In parallel, crowdsourced geodata is rapidly emerging, and projects like OpenStreetMap [29], GeoNames [16], or Wikipedia [46] currently offer reliable, up-to-date geographic information for free.

Besides, knowledge representation and reasoning according to the Linked Data paradigm [5] is extremely useful in Semantic Web

applications, such as online shopping platforms, personalized content delivery, etc. Crowdsourced initiatives like DBpedia [7] extract structured information from Wikipedia [46] and link it to other web resources. Thanks to knowledge representation models like RDF [35] and OWL [32], and query protocols such as SPARQL [39], much work is done towards transforming relational data into RDF, including standardization by the RDB2RDF W3C group [36].

Linked data technologies can also provide the means for semantic manipulation of spatial features, including interlinking, querying, reasoning, aggregation, fusion, and visualisation. With a handful of notable exceptions [2, 31, 43], only a small amount of such information has been published as *linked geospatial data* and associated with other resources in the Semantic Web. A major difficulty has to do with the inherent complexity of geospatial concepts. Apart from points, which can be simply abstracted as a pair of latitude/longitude coordinates, all other geometries require more robust representations to cope with irregular shapes (e.g., polygons with holes, or curves with multiple disconnected parts). It is also difficult to express implicit topological relationships between web resources, e.g., to provide an answer to the query "Find all subway stations within 1 km distance from my hotel". After several initial proposals, the recent OGC GeoSPARQL standard [25] suggests a unified approach for representing linked geospatial data as RDF triples and querying them through a SPARQL extension equipped with a variety of spatial operators and functions [4]. Yet, it is surprising the lack of tools for converting geospatial features from several sources into GeoSPARQL-compliant serializations.

Towards this goal, we introduce open-source utility TripleGeo [1]. Our aim is to bridge the gap between typical geographic representations from a variety of proprietary files, DBMSs, and georeference systems with the demands of geospatially-enabled RDF stores. Development was based on open-source `geometry2rdf` library [19], but with notable modifications and substantial enhancements to meet interoperability needs in RDF stores. In fact, TripleGeo is designed as a spatial ETL tool, enabling users to: (i) *Extract* spatial data from a source; (ii) *Transform* this data into a triple format and geometry vocabulary prescribed by the target RDF store; and (iii) *Load* resulting triples into the target RDF store. Therefore, TripleGeo always preserves data integrity and provides consistent, well-defined geospatial information to end users.

Among its distinctive features, we point out that TripleGeo can:

- Directly access de facto geographic formats (e.g., shapefiles [9] or DBMSs (e.g., Oracle Spatial [30] or PostGIS [33]).
- Recognize many *geometric data types*, i.e., not only points, but (multi-)linestrings and (multi-)polygons as well.
- Extract *thematic attributes*, e.g., identifiers, names, or types, associated with each feature.

- Allow on-the-fly reprojection between *Coordinate Reference Systems* (CRS), e.g., transform geometries from GreekGrid87 (a local CRS) into WGS84 (used for GPS locations).
- Export triples into various *notations* (RDF/XML, TTL, etc.) and geometry vocabularies for swift loading into RDF stores.

The remainder of this paper proceeds as follows. In Section 2, we survey related work on specifications and tools for converting relational and geospatial data into RDF. In Section 3, we present TripleGeo's architecture, by examining its components and processing flow, along with its dependencies on third-party platforms and libraries. In Section 4, we discuss the current implementation status and planned extensions for future releases of TripleGeo.

## 2. RELATED WORK

Creating knowledge from structured (e.g., relational databases, XML) or unstructured sources (e.g., text, images) can be extremely valuable in the Semantic Web. The R2RML Recommendation [37] by W3C specifies an RDF notation for mapping relational tables, views or queries into the RDF data model. Among the thirty tools for knowledge extraction reviewed in [42], the majority are considered as proof-of-concept prototypes. Some of them are rather "mature" tools for transforming relational databases into RDF, such as Triplify [3], D2R Server [8], or Virtuoso's RDFizer Middleware (Sponger) [27]. During conversion, these tools allow reuse of existing vocabularies and ontologies. Although under development, the Google Refine RDF Extension [38] seems promising, and can reconcile against SPARQL endpoints and RDF dumps. However, none of the aforementioned methodologies and tools currently provides any particular support for geospatial data and operations.

On the other hand, several ETL tools can manage the unique characteristics of spatial data. Among them, *GDAL/OGR* [13] is an open-source translator library implementing the OGC vector model [24] and can handle proprietary storage models for many geospatial DBMSs and files. *GeoKettle* [40] is a metadata-driven spatial ETL tool dedicated to integration of different data sources for building and updating geospatial data warehouses. Finally, *FME Workbench* is included in ESRI's ArcGIS Data Interoperability extension [10] and enables transformation of geometric and thematic attributes along with schema redefinitions. Currently, such utilities are mainly used for data cleaning, merging, verification or conversion into various formats, but have absolutely no RDF support.

There have been several proposals for geospatial RDF data management such as [14, 17, 47], but none provided a solid framework for developing large-scale applications and services. Recently (2012), the OGC GeoSPARQL standard [25] suggests a concrete ontology for representing features and geometries in RDF as Well Known Text (WKT) or Geography Markup Language (GML) literals. GeoSPARQL defines a core set of classes, properties and data types that can be used to construct query patterns in an extension of SPARQL. To cope with incompatible methods for representing and querying spatial data, GeoSPARQL follows other OGC standards [24]. With such standardization, both vendors and users can achieve uniform, transparent, platform-independent access to geospatial RDF data with a rich collection of query operators. Currently, only few RDF stores like Parliament [34] or uSeekM [28] have partially implemented GeoSPARQL specifications. Instead, several geospatially-enabled triple stores prefer proprietary geometric representations (e.g., AllegroGraph [12]) or restrict their support to points only (such as OWLIM [23] or Virtuoso [26]).

To the best of our knowledge, there have been very few initiatives specifically for converting geospatial features into RDF resources. Data conversion into an appropriate RDF format using

a selected ontology is among the functionalities supported by the generic *DataLift* platform [6]. Although geometries can be extracted as WKT strings under a custom namespace, there is currently no support for GeoSPARQL. *LinkedGeoData* [2] aims at adding a spatial dimension to the Semantic Web. It offers a flexible platform for mapping OpenStreetMap (OSM) data [29] to RDF, a SPARQL endpoint for making RDF data publicly available, as well as several tools for performing mappings and interlinking of geospatial semantic data. The resulting graph comprises more than 20 billion triples interlinked with DBpedia [7] and GeoNames [16]. Nevertheless, spatial operations deal strictly with OSM nodes and ways, ignoring any other geographic sources or data types.

In parallel, *Geo.LinkData.es* is an open initiative to enrich the Web of Data with geospatial data for Spain. Among the tools they developed, *geometry2rdf* [19] enables extraction of geometries as RDF triples [44]. Geometries can be available in GML or WKT serializations and are manipulated with GeoTools [18], not only in order to retrieve features, but also to perform coordinate transformation. However, its RDF model is not compliant with the GeoSPARQL standard [25], and cannot handle attribute values (e.g., name literals) or export triples in various notations apart from RDF/XML. Concerning interaction with geospatial sources, it only supports geometry extraction from shapefiles [9] and Oracle Spatial [30]. Despite these important deficiencies, this open source library provided an initial base for developing our own utility TripleGeo. As we explain next, we particularly aim at integrating several external databases and providing support for GeoSPARQL data types.

## 3. CONVERTING GEOMETRIES TO RDF

In this Section, we present the architecture and capabilities of ETL tool TripleGeo for converting vector geospatial features into RDF triples. This process iterates through all features in the input dataset and emits a series of triples per record. Every geometric feature is converted into properly formatted triple(s), according to the specified RDF vocabulary. Most typically, geometries are turned into WKT serializations as prescribed by GeoSPARQL [25], but some legacy namespaces are supported as well. Thematic attributes can be extracted in tandem, such as identifiers, names, or classifications. Results are written into a file using standard triple notations, so that they can be readily loaded into an RDF store.

### 3.1 Integrated Libraries

TripleGeo inherits from *geometry2rdf* dependencies to various open-source tools and libraries, all of which are used "as is". The most significant of these libraries are:

- *Apache Jena* [21] is a widely used Java framework for developing Semantic Web applications, tools and servers.
- *GeoTools* [18] offer Java implementations of OGC specifications [24] for geospatial data management comparable to GIS desktop applications and web services. Its rich API supports feature access to many file formats (like CSV, DXF, GeoJSON, ESRI shapefiles, etc.) and spatial DBMSs, as well as coordinate transformations between CRS.
- *GDAL/OGR*. We actually make use of OGR Simple Features embedded in this Geospatial Data Abstraction Library [13]. This includes command-line tools for read access to a variety of *vector* formats (shapefiles, PostGIS, Oracle Spatial, etc.).
- *Java Topology Suite (JTS)* [45] is an open source API that provides support for 2-dimensional topological predicates and spatial functions conforming to OGC [24].



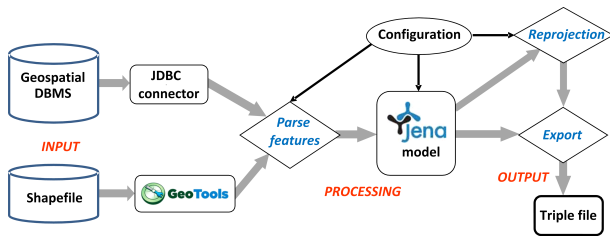


Figure 1: Processing flow diagram for ETL utility TripleGeo.

### 3.2 TripleGeo Components

TripleGeo has been implemented with several Java classes that perform specific tasks in a modular fashion. From a user’s perspective, this command-line utility is entirely automated according to preconfigured settings. Figure 1 illustrates the flow diagram used for converting geospatial features into RDF triples. Next, we outline the basic components of the utility:

- *Input* data may be obtained either from geographic files or geospatially-enabled DBMSs, as explained in Section 3.3.
- *Connectors* to source data are required in order to access geometric features. In case of a DBMS, this is possible thanks to suitable JDBC drivers. With respect to file formats, the integrated GeoTools library provides all required functionality.
- A *configuration file* declares user preferences concerning all stages of the conversion: how input source will be accessed, which data is involved, what geometric representation should be used, whether geometries must be reprojected into another CRS, as well as the output triple notation.
- A *parser* iterates through each input record and converts geometries into a suitable representation according to user specifications. It also consumes thematic attribute values (e.g., types, names) and emits properly formatted literals.
- A *Jena model* is a main-memory data structure that is used to retain all state information consisting of the collection of generated triples. This model denotes an RDF graph, so called because it contains a collection of RDF nodes, attached to each other by labelled relations. In Java terms, this model acts as the primary container of RDF information in graph form. A significant benefit from using the Jena model is that it offers a rich API with many methods intended to make it easier to write RDF-based applications.
- Optionally, *reprojection* of geometries into another spatial reference system is possible. This transformation is carried out thanks to the integrated GeoTools library and according to user specifications for the source and target CRS.
- *Export* of generated triples into a single file is performed by the Jena API. This offers the possibility of writing the output into several triple formats, as discussed in Section 3.4.

### 3.3 Input

Current version 1.0 of TripleGeo can access geometries from:

- ESRI shapefiles [9], which is a well-known format for storing geospatial features in files.
- Geospatially-enabled DBMSs, such as: IBM DB2 with Spatial Extender [20], MySQL [22], Oracle Spatial and Graph [30], and PostGIS (spatial module for PostgreSQL) [33].

Geometric data must reside in a single table (in case of a database) or a file. Combining thematic information from multiple tables (e.g., via joins) from the same source is also available. However, it is not currently possible to concurrently process data from diverse sources or formats. Attributes (i.e., table columns) that can be extracted from a given data source include:

- The *geometry* itself (mandatory), expecting valid, georeferenced points, (multi-)linestrings, and (multi-)polygons according to the OGC specification [24].
- A unique *identifier* (mandatory) for each entity, which will be used to generate the IRI of the extracted resource.
- Optionally, a *name* value associated with an entity can be converted into a string literal.
- Optionally, a *type* value that characterizes an entity can be associated with an `rdf:type` predicate.

### 3.4 Output

In terms of output serializations, and according to the specifications of the Jena API [21] that exports the model, the triples can be obtained in one of the following notations: *RDF/XML* (default), *RDF/XML-ABBREV*, *N-TRIPLES*, *N3*, and *TURTLE (TTL)*.

In terms of standardization, output triples are conformant to W3C standards, thanks to Jena API methods for creating resources, properties and literals, and statements linking them. Therefore, all output triples are compatible with the most commonly used standards, including RDF, RDFS, OWL, and SPARQL.

With respect to geospatial features, triples can be exported according to the GeoSPARQL standard [25]. TripleGeo also supports legacy namespaces, such as `pos:` of the W3C Basic Geo Vocabulary [47] or Virtuoso’s `virttrdf:` for custom point geometries [26]. But note that such syntaxes are neither compliant to GeoSPARQL nor can they handle shapes other than points.

Basically, the output geometry serialization depends on the RDF store where triples will be loaded afterwards. Parliament [34] and uSeekM [28] only accept GeoSPARQL-compliant triples. Virtuoso [26] requires its own custom syntax and currently handles point features only. OWLIM [23] supports only points under the W3C Basic Geo Vocabulary. Other RDF stores like Oracle [30] or Strabon [41] are close, but not fully conformant to GeoSPARQL, mainly due to differing namespaces. In that case, geometries can be extracted into GeoSPARQL and then replace the necessary prefixes.

## 4. CURRENT STATUS AND OUTLOOK

TripleGeo is free software and its current version 1.0 is available from [1], including the Java source code and sample data. We provide distributions in both platform-neutral Java JARs and Debian-specific DEB packages. TripleGeo can be redistributed or modified under the terms of the GNU General Public License. This tool is also integrated into `stack.linkeddata.org`, which comprises many utilities for managing the lifecycle of Linked Data.

We have tested TripleGeo with diverse input formats, RDF stores, data sources, and geometric serializations. In one test case scenario, OpenStreetMap layers [29] for Great Britain were converted into more than 25 million triples, including 3.5 million geometries (points, polylines, and polygons). TripleGeo can readily accept shapefiles from OSM dumps as input and convert them into RDF triples. However, these original shapefiles were also imported into databases hosted in PostGIS [33] and Oracle Spatial [30]. Thus, we have also conducted ETL operations from these spatial DBMSs

into triples, and we were able to verify that TripleGeo can also interact and access spatial features from major DBMSs. Apart from verifying its functionality, we also performed some more comprehensive tests by converting large datasets into triples; a detailed evaluation is available at [15]. Not only has such testing proven the robustness of the tool, but the differing geospatial specifications of each RDF store also guided development and progressive refinement towards handling as many cases as possible.

TripleGeo is still a work-in-progress. Thanks to its modular implementation, more utilities are under development without affecting existing functionality, including interaction with more geographic data sources (e.g., GML, KML, etc.) and DBMS platforms, as well as support for more complex geometric types (e.g., geometry collections [24]). We also plan to expose the full functionality of TripleGeo via a RESTful API (e.g. for web-accessible data), and also offer a web interface to upload, convert and download large datasets. As scalability with increasing data volumes is most challenging, a possible solution would be to automatically split the input into disjoint batches and use a parallelization scheme like MapReduce to generate triples. Last, but not least, ability to define mappings and vocabularies and export geometric and thematic values under a user-specified ontology would be noteworthy. As a test case, we have begun developing an ETL methodology for INSPIRE-compliant [11] data and metadata.

## 5. ACKNOWLEDGEMENTS

This work was partially supported by the European Commission under EU/FP7 grant #318159 for project "*GeoKnow: Making the Web an Exploratory Place for Geospatial Knowledge*".

## 6. REFERENCES

- [1] Athena R.C. TripleGeo open source utility. URL: <https://github.com/GeoKnow/TripleGeo>
- [2] LinkedGeoData project. URL: <http://linkedgeo.org>
- [3] S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller. Triplify: Light-weight linked data publication from relational databases. In *WWW*, pp. 621-630, April 2009.
- [4] R. Battle and D. Kolas. GeoSPARQL: Enabling a Geospatial Semantic Web. *Semantic Web Journal*, 3(4):355-370, 2012.
- [5] C. Bizer, T. Heath, and T. Berners-Lee. Linked Data - The Story So Far. *IJSWIS*, 5(3): 1-22, 2009.
- [6] DataLift project. URL: <http://datalift.org/>
- [7] DBpedia. URL: <http://dbpedia.org>
- [8] D2R Server. URL: <http://d2rq.org/d2r-server>
- [9] ESRI Inc. Shapefile Technical Description. URL: <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>
- [10] ESRI Inc. FME Workbench for ArcGIS Data Interoperability URL: <http://www.esri.com/software/arcgis/extensions/datainteroperability/key-features/spatial-etl>
- [11] European Commission. INSPIRE Directive – Infrastructure for Spatial Information in the European Community. URL: <http://inspire.jrc.ec.europa.eu/>
- [12] Franz Inc. AllegroGraph Triple Store. URL: <http://www.franz.com/agraph/allegrograph/>
- [13] GDAL/OGR library. URL: <http://www.gdal.org/>
- [14] GeoJSON 1.0. URL: <http://geojson.org/>
- [15] GeoKnow Deliverable D2.2.1: Integration of External Geospatial Databases. URL: <http://geoknow.eu/t2-2.html>
- [16] GeoNames database. URL: <http://www.geonames.org/>
- [17] GeoRDF Profile. URL: <http://www.w3.org/wiki/GeoRDF>
- [18] GeoTools library. URL: <http://www.geotools.org/>
- [19] GeoLinkedData.es Team. geometry2rdf Utility. URL: <https://github.com/boricles/geometry2rdf>
- [20] IBM DB2 Spatial Extender. URL: <http://www.ibm.com/software/products/us/en/db2spaext/>
- [21] Apache Jena project. URL: <http://jena.sourceforge.net/>
- [22] MySQL Database. URL: <http://www.mysql.com/>
- [23] Ontotext AD. OWLIM Semantic Repositories. URL: <http://www.ontotext.com/owlim>
- [24] OGC Inc. Implementation Specification for Geographic Information - Simple Feature Access. URL: [http://portal.opengeospatial.org/files/?artifact\\_id=25354](http://portal.opengeospatial.org/files/?artifact_id=25354)
- [25] OGC Inc. GeoSPARQL Standard - A Geographic Query Language for RDF Data. URL: [https://portal.opengeospatial.org/files/?artifact\\_id=47664](https://portal.opengeospatial.org/files/?artifact_id=47664)
- [26] OpenLink Software. Virtuoso Universal Server. URL: <http://virtuoso.openlinksw.com/>
- [27] OpenLink Software. Virtuoso's RDFizer Middleware (Sponger). URL: <http://docs.openlinksw.com/virtuoso/virtuososponger.html>
- [28] OpenSahara uSeekM library. URL: <https://dev.opensahara.com/projects/useekm/>
- [29] OpenStreetMap project. URL: <http://www.openstreetmap.org/>
- [30] Oracle Inc. Oracle 12c Spatial and Graph. URL: <http://www.oracle.com/technology/products/spatial>
- [31] Ordnance Survey. Linked Data Platform. URL: <http://data.ordnancesurvey.co.uk/>
- [32] OWL Web Ontology Language: <http://www.w3.org/TR/owl>
- [33] PostGIS - Spatial and Geographic Objects for PostgreSQL. URL: <http://postgis.net/>
- [34] Raytheon BBN Technologies Inc. Parliament Triple Store. URL: <http://parliament.semwebcentral.org/>
- [35] Resource Description Framework Primer. URL: <http://www.w3.org/TR/rdf-primer/>
- [36] RDB2RDF Working Group. URL: <http://www.w3.org/2001/sw/rdb2rdf/>
- [37] R2RML: RDB to RDF Mapping Language. URL: <http://www.w3.org/TR/r2rml/>
- [38] RDF Refine: a Google Refine extension for exporting RDF. URL: <http://refine.deri.ie/>
- [39] SPARQL 1.1 Query Language for RDF. URL: <http://www.w3.org/TR/sparql11-query/>
- [40] Spatialytics.org. GeoKettle Spatial ETL tool. URL: <http://www.spatialytics.org/projects/geokettle/>
- [41] TELEIOS EU/FP7 project. Strabon prototype. URL: <http://strabon.di.uoa.gr/>
- [42] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge Extraction from Structured Sources. In *Search Computing III*, pp. 34-52, 2012.
- [43] U.S. Geological Survey. Building Ontology for the National Map. [http://cegis.usgs.gov/ontology\\_userguide.html](http://cegis.usgs.gov/ontology_userguide.html)
- [44] L.M. Vilches-Blázquez, B. Villazón-Terrazas, V. Saquicela, A. de León, O. Corcho, and A. Gómez-Pérez. GeoLinked Data and INSPIRE through an Application Case. In *ACM SIGSPATIAL GIS*, pp. 446-449, November 2010.
- [45] Vivid Solutions Inc. JTS Topology Suite. URL: <http://www.vividsolutions.com/jts/JTSHome.htm>
- [46] Wikipedia. URL: <http://wikipedia.org>
- [47] W3C Basic Geo (WGS84 lat/long) Vocabulary. URL: <http://www.w3.org/2003/01/geo/>

# Multimodal Social Data Management (MSDM)

Antonio Penta (University of Turin, Italy)  
Claudio Schifanella (University of Turin, Italy)  
Carlos Ruiz (playence GmbH, Austria)  
Maria Luisa Sapino (University of Turin, Italy)

# Social Data and Multimedia Analytics for News and Events Applications

Ioannis Kompatsiaris  
Information Technologies Institute  
6th Km Charilaou-Thermi road  
GR57001, Thessaloniki, Greece  
+30 2311257774  
ikom@iti.gr

Sotiris Diplaris  
Information Technologies Institute  
6th Km Charilaou-Thermi road  
GR57001, Thessaloniki, Greece  
+30 2311257778  
diplaris@iti.gr

Symeon Papadopoulos  
Information Technologies Institute  
6th Km Charilaou-Thermi road  
GR57001 Thessaloniki, Greece  
+30 2311257772  
papadop@iti.gr

## ABSTRACT

This paper discusses a framework enabling real-time multimedia indexing and search across multiple social media sources. It places particular emphasis on the real-time, social and contextual nature of content and information consumption in order to integrate topic and event detection, mining, search and retrieval, based on aggregation and indexing of shared user-generated multimedia content. User-friendly applications for the News and Events domains have been developed based on these approaches, incorporating novel user-centric media visualisation and browsing methods. The research and development is part of the FP7 EU project SocialSensor.

## Categories and Subject Descriptors

I.5.4 [Pattern Recognition]: Applications; H.3.3 [Information Search and Retrieval]: Retrieval models

## General Terms

Algorithms, Theory, Experimentation

## Keywords

social media, multimedia analysis, multimedia search, analytics

## 1. INTRODUCTION

Social networks have become an integral part of modern life driving more and faster communication than ever before. Media content is created and published online at unprecedented rates by both regular users and professional organisations. At the same time the wide availability of smartphones has enabled the creation and instant sharing of media content at the time and place of an event, creating an “online reflection” of what happens in the real world. However, the fast pace, the huge volume and the unpredictable nature of user-contributed content make it extremely challenging to obtain informative views of evolving news stories and events in real time and to quickly surface relevant media content.

Especially with regard to the news industry the challenge is to use and embrace new content authoring and provision methods, and the channels offered by social media and mobile technologies, to

their fullest advantage, for both information gathering and information distribution. A key challenge in this respect is to develop appropriate tools for quickly surfacing trends, sentiments and discussions in social media, in relevant and useful ways.

Another domain where social media analytics are gaining great importance is the organization of large events, such as festivals and expos. As attendants of such events need to organise their visits, new methods providing context-aware information are becoming necessary in order to enhance user experience. Event organisers can also benefit from social media sensing applications that can help them capture the pulse of their events and gain valuable insights into their impact on visitors. To this end, they need tools that help them make sense of the large amounts of online messages and shared content.

This short paper presents a framework for social media analytics that have been developed in the context of SocialSensor, a 3-year FP7 European Integrated Project<sup>1</sup> aiming to tackle some of the challenges outlined above and to offer solutions as well as improvements in the industry domains of professional news and event organisation. In the first domain, we present a system for crawling, indexing and browsing social media content with the goal of facilitating the discovery of newsworthy and interesting social multimedia. In the second, we present the EventSense application that helps event organisers extract insights from large events by mining large amounts of online messages shared through OSNs. Offering real-time social indexing capabilities for both of these use cases is expected to have a transformational impact on both sectors.

The subsequent chapters outline some of the research applications developed by SocialSensor, for sensor mining and social search, as well as results from their evaluation.

## 2. SOCIAL MULTIMEDIA SEARCH

The first application facilitates the targeted collection, indexing and browsing of shared media content through a hybrid crawling strategy, comprising both a stream-based and a query-driven approach. In addition, it integrates very efficient and scalable image indexing and clustering implementations.

The crawler is responsible for the collection of data and content from online sources in the form of Items (posts made in a social platform, e.g. tweets), WebPages (URLs embedded in collected Items) and MediaItems (images/videos embedded in Items or WebPages), given a set of crawling specifications (arguments

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

<sup>1</sup> <http://www.socialsensor.eu>

specifying what to crawl, e.g. a hashtag on Twitter) as input. The proposed crawling and indexing framework, depicted in Figure 1, comprises the following components: (a) Item collection, comprising stream-based (stream-manager) and query-driven (search-manager) components, (b) structured data repositories based on mongoDB and Solr, (c) fetching and indexing components, including WebPage fetching, article extraction, MediaItem detection and extraction, feature extraction and visual indexing [1], (d) aggregation components, including geo, visual clustering and analytics. Several of the system components are available as open-source projects on GitHub<sup>2</sup>, intended for use in the professional journalism domain [2].

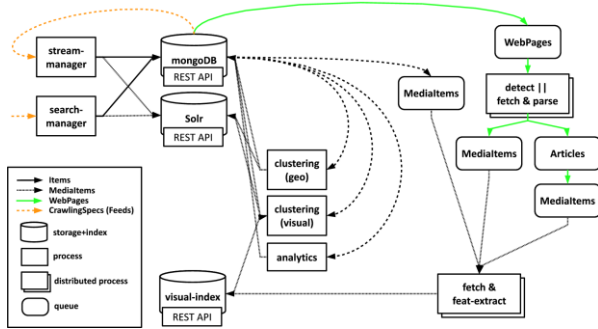


Figure 1. Social multimedia crawling and search.

### 3. SOCIAL MEDIA SENSING

The EventSense application enables a series of mining operations on social media content around large events, including the automatic association of online messages to entities of interest (e.g. films in the case of a film festival), the discovery of trending topics, and the detection of sentiment (positive, negative, neutral) both at an entity level (e.g. per film) and on aggregate. In addition, the application produces an informative social media summary of the event of interest by automatically selecting and putting together its highlights, e.g. the most discussed entities and topics, the most influential users, the evolution of the discussions' sentiment, and the most shared media and news content.

More specifically, online messages about the event are organized around entities of interest (e.g. films) and topics, and sentiment scores are extracted for each of those, by aggregating the sentiment expressed by individual messages. An overview of the EventSense application is illustrated in Figure 2. For more details please see [3].

### 4. RESULTS

With respect to the social multimedia crawling and search framework we evaluated its media retrieval capabilities in the context of the #OccupyGezi events. We selected images related to the event from four different sources and used them as visual queries to the NN search component. To assess the search performance, we computed a *coverage* score by dividing the number of queries with at least one correct result with the total number of queries. We note that in most cases coverage ranges from 0.5 to 0.95. Regarding the quality of ranking for similar images, in most cases where at least one correct result was found to be the same or very similar to the query image, the correct

results were ranked above the incorrect ones. It is also noteworthy that in all cases, the response times were sub-second, typically in the range of 100-200msec. More details are available in [2].

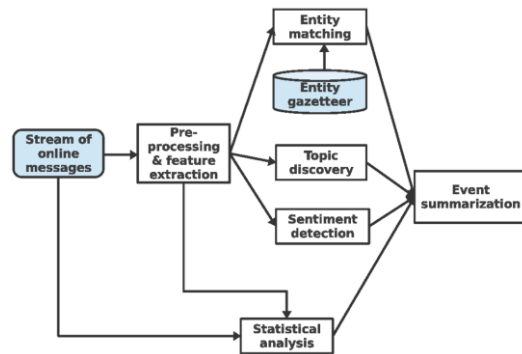


Figure 2. EventSense system.

We conducted an evaluation of EventSense on a Twitter dataset around the 53rd International Film Festival of Thessaloniki taking place between November 2nd and 11th, 2012. The evaluation results provided evidence that real-world event variables, such as film ratings, are correlated with aggregate statistics mined from the stream of online messages. For further details, please see [3].

## 5. CONCLUSION

We presented two applications for real-time social media content indexing, search and retrieval tailored around the needs of news professionals, as well as organisers and attendees of large events. To this end, innovative analysis techniques of social data and content, assisted by effective indexing of real-time social media streams were developed. Evaluation studies demonstrate the effectiveness of such techniques in collecting diverse content from social networks, in browsing and searching it in multiple ways, and in perceiving the pulse of large-scale events.

## 6. ACKNOWLEDGMENTS

This work is supported by the SocialSensor FP7 project, partially funded by the EC under contract number 287975.

## 7. REFERENCES

- [1] Spyromitros-Xioufis, E., Papadopoulos, S., Kompatsiaris, I., Tsoumakas, G., & Vlahavas, I. (2012, May). An empirical study on the combination of surf features with VLAD vectors for image search. In *Image Analysis for Multimedia Interactive Services (WIAMIS), 2012 13th International Workshop on* (pp. 1-4). IEEE.
- [2] Papadopoulos, S., Schinas, E., Mironidis, T., Iliakopoulou, K., Spyromitros-Xioufis, E., Tsampoulatis, I., and Kompatsiaris, I., 2013. *Social Multimedia Crawling and Search*. In *IEEE Computer Society Special Technical Community on Social Networking E-Letter*, vol. 1, no. 3, October 2013.
- [3] Schinas, E., Papadopoulos, S., Diplaris, S., Kompatsiaris, I., Mass, Y., Herzig, J., and Boudakidis, L. 2013. *EventSense: capturing the pulse of large-scale events by mining social media streams*. In *Proc. 17th Panhellenic Conference on Informatics (PCI '13)*. ACM, New York, NY, USA, 17-24.

<sup>2</sup> <https://github.com/socialsensor>

# Event Identification and Tracking in Social Media Streaming Data

Andreas Weiler, Michael Grossniklaus, and Marc H. Scholl  
University of Konstanz  
Dept. of Computer & Information Science  
Box D 188, 78457 Konstanz, Germany  
firstname.lastname@uni-konstanz.de

## ABSTRACT

In recent years, the growing popularity and active use of social media services on the web have resulted in massive amounts of user-generated data. With these data available, there is also an increasing interest in analyzing it and to extract information from it. Since social media analysis is concerned with investigating current events around the world, there is a strong emphasis on identifying these events as quickly as possible, ideally in real-time. In order to scale with the rapidly increasing volume of social media data, we propose to explore very simple event identification mechanisms, rather than applying the more complex approaches that have been proposed in the literature. In this paper, we present a first investigation along this motivation. We discuss a simple sliding window model, which uses shifts in the inverse document frequency (IDF) to capture trending terms as well as to track the evolution and the context around events. Further, we present an initial experimental evaluation of the results that we obtained by analyzing real-world data streams from Twitter.

## Categories and Subject Descriptors

H.3 [Information Storage and Retrieval]: Information Storage and Retrieval; H.4 [Information Systems Applications]: Miscellaneous

## Keywords

event detection, stream processing, social media analytics

## 1. INTRODUCTION AND MOTIVATION

The continuous emergence of new web services, such as social media platforms and technologies for generating and receiving streamed data, imposes new challenges on the way such data volumes are processed and analyzed in real-time or near real-time. Since the users of information services are typically interested in current events and actual happenings

of the world, it is necessary to retain the real-time characteristic of the streams and to perform the identification of real-world events as fast as possible.

In this paper, we focus on the use-case of Twitter, the most popular social microblogging site, which produces a large volume of data as a continuous stream of messages, so-called “tweets”. Since its inception, the way people use Twitter has undergone a remarkable evolution. While Twitter was initially intended as a service to share short personal status messages, it quickly became a platform that people used to report on and stay informed about current events happening all around the world. This change in usage was also reflected in Twitter’s user interface by changing the original prompt “*What are you doing?*” to the more general question “*What’s happening?*”<sup>1</sup>.

Another important characteristic of Twitter is its vibrant community with, as of 2013, 200 million daily active users from all around the world. Due to its lightweight approach to broadcasting, important news rapidly propagate through Twitter’s densely interconnected social network. Although the resulting volume and variety of content in the information flow is a great opportunity for data analysis, it also gives rise to the challenge of detecting significant messages that can be used to identify events in the frenzy of tweets. As a consequence, several state of the art approaches exist that address the problem of accurate event detection. In the last six years, tweets have gone from 16 millions per year to 400 millions per day. Taking this rapid growth into consideration, we believe that the scalability of event identification is as important as its accuracy. Therefore, we propose to study simple approaches that, ideally, can be tuned to trade-off precision for lower computational complexity.

In this paper, we present a first exploration into this direction. We propose a simple event identification approach, which uses a sliding window model to extract events and the context of events in real-time from the live public data stream of Twitter. Our approach is based on monitoring shifts in the inverse document frequency (IDF) of terms and therefore suggests that it is possible to handle large amounts of data and get important insights by means of aggregation only. Since our approach is based on windowing, the window size is a natural parameter that can be used to control the precision/complexity trade-off. Apart from the approach itself, we also present a first evaluation based on a case study to obtain an indication of the results that can be expected from such an approach.

<sup>1</sup><http://blog.twitter.com/2009/11/whats-happening.html>

## 2. EVENT ANALYSIS

*Event detection* is a classical problem in computer science and has been studied for many years in various research areas. A lot of research is dedicated to detecting anomalies or novelties in different data sources ([5, 11]), considering those phenomena to be an indication of an event. Further related research deals with the detection of changes or drifts in data streams ([1, 7, 9]).

Taking into account a vast number of Twitter messages generated each second, it becomes a crucial task to group messages by topics or events. Because there is no explicit knowledge about current or future events, the latter have to be identified and detected in an on-line fashion without limitation to any domain via predefined keywords. We propose that only by means of aggregation it is possible to handle large amount of data and gain important insights into it. Therefore, the detected high-level representations can be used to compress the tweets in a meaningful manner.

Our approach of using sliding windows to identify events in streaming data considers the timestamp information included within the tweets as a basis for the window sizes. In the following, we describe the identification of events by using textual analysis of the content of tweets and statistical analysis of the frequency of terms. Once an event has been detected, we keep track of the most co-occurring terms with the event term, which mainly describe the context around an event. This can be helpful to provide a better overview and more insights into the event’s evolution to analysts and other users.

### 2.1 Event Identification

The content of the tweet messages provides a high variety of information and as such can be considered as the most important dimension in the data set. In the following, we describe the process of event identification by analyzing the content of tweets.

The first step of the event term extraction process is the tokenization and part-of-speech tagging of the tokens by using a especially for Twitter tailored tokenizer and Part-Of-Speech Tagger [8]. Since pure tokenization of texts results in an abundance of terms, but the main subject of an event is typically reflected in nouns we filter the resulting set of tokens to nouns and proper nouns for further treatment. Additionally to single nouns, we also take bigrams (nouns with preceding adjectives, verbs, or nouns) into account. We further reduce the token set by discarding all tokens, which are shorter than four characters, as well as tokens contained in a standard English stopword list, or containing any non-alphabetic characters. Because the spelling of words in social media can be very diverse and the amount of terms would increase a lot, we also discard tokens with more than three successive repetitions of the same character (e.g., “hellooooo”, “goooooaalll”). Once the preprocessing is done, each tweet message is represented by a set of terms  $T = (t_1, t_2, t_3, \dots)$ . To avoid wrong identification of event candidates by continuous repetition of the same term, these sets are kept duplicate-free.

The evolution of relevant terms is evaluated by an ongoing process using the following rules. The incoming stream  $TWS = (tw_1, tw_2, tw_3, \dots)$  is partitioned into fixed sized windows  $(w_1, w_2, w_3, \dots)$ . For each extracted term (we call them event candidates  $ec$ ) in the stream we continuously calculate an *IDF* [16] ( $idf(ec)$ ) value for each of the windows. In ad-

dition, we calculate the percentage of the shift of the *IDF* value from one window to another ( $sidf(ec)$ ), which is only possible if the  $ec$  occurs in two successive windows. For further evaluation, we also calculate the average *IDF* value ( $avg(idf(ec))$ ) of all terms in the window and the average value of all shifts  $avg(sidf(ec))$  (possible for all  $ec$  occurring in the last two successive windows) between two succeeding windows.

If the  $ec$  occurs in more than  $n$  successive windows, we check the *IDF* value of all  $n$  windows against the average value  $avg(idf(ec))$  of the corresponding window. If all values are lower then the average value the  $ec$  is further evaluated. After the first check, we check the *IDF* value shift  $sidf(ec)$  for the windows  $(w_{n-3}, w_{n-2})$ ,  $(w_{n-2}, w_{n-1})$  and  $(w_{n-1}, w_n)$  against the corresponding average values. If all shift values are higher than the corresponding average value and the added up shift value is over a certain threshold, we identify this  $ec$  as an event term. In this way, both fast and slower increasing event terms can be identified, and the identification of event terms adapts to dynamically changing boundaries, which are the currently existing average values.

Since the amount of event candidates in the term set increases continuously we discard all terms, which are missing in a window. Figure 3 shows that there is an almost equivalent number of terms in the windows over time. Event candidates, which are identified as event terms are passed on to the event tracking phase, which is described in the next section.

### 2.2 Event Tracking

Once the event identification phase has identified an event candidate as event term, it is also interesting for an analyst to keep track of the event, to get an ongoing overview and insight of the happenings related to the event, or to evaluate the importance of an event. Therefore, the event tracking phase of our analysis is initiated after an event term is identified.

To support this process, we extract all co-occurring terms of the event term, which includes verbs, nouns, and adjectives and use the term cleaning methods, which are mentioned before. Afterwards we calculate the percentage of the co-occurrences of the term with the event term in the corresponding window and order them by the percentage value. To summarize the context around a identified event term the top  $n$  co-occurrence terms are extracted continuously.

## 3. EVALUATION

In this section, we describe the evaluation of our approach in terms of the experimental setup and the experiments that we conducted. In both experiments, we applied an implementation of our approach to the identification of events in the Twitter social data stream and subsequent tracking of the evolution of these identified events.

### 3.1 Setup

The Twitter platform provides direct access to the public live stream of Twitter messages via a set of developer APIs. The Twitter API [17] enables application developers to receive a large portion of the total number of daily produced tweets. By using the Twitter Streaming API with the so-called “gardenhose” access level, we are able to collect 10% of the total public live stream. Additionally, we merge our data set with a geo-filtered stream to increase the number



of geo-tagged tweets. We can assume that about 10% of the incoming tweets have geographic information available. This information is set either automatically by the mobile device, or manually by the author of the tweet, or by both of them. Figure 1 displays statistics about the amount of incoming tweets for a representative sample of days. From these numbers we can conclude that we can receive an average of over one million tweets per hour with the average of 20,000 tweets per minute. Further we can see that there is a certain decrease in the number of tweets between the hours 11 and 16 each day.

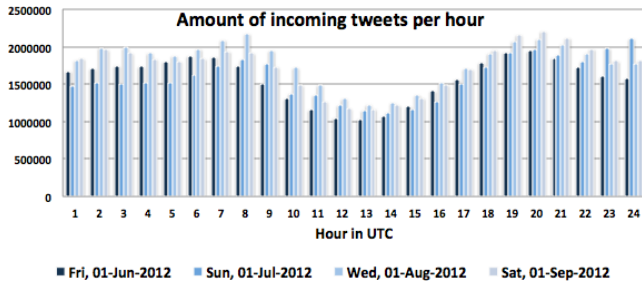


Figure 1: Number of incoming tweets per hour for the first day of the months June, July, August, and September 2012.

In our implementation, we rely on the native XML database system *BaseX*<sup>2</sup> for data management and processing. All designated incoming data is stored in a standardized format to support fast and easy data access. The data in the Twitter streams is in the JavaScript Object Notation (*JSON*) format, which is efficiently converted to XML on-the-fly using the JSON conversion functionality of *BaseX*. Since this solution converts the JSON object directly to XML, there is an automatic adaptation to all potential changes in the format of the streamed Twitter data.

For this work, we simulate a live-stream behavior of Twitter by pushing stored tweets from the database in the same sequence, in which they were gathered from the stream. The client can register a filter query—formulated in *XQuery*—on the stream to receive only a requested type of tweets from the database system. For example, it is possible to filter for tweets with valid geographic information or for tweets containing a specified keyword. By simulating a live streaming environment we ensure that the analysis can also be directly applied to the on-line stream by connecting to the live Twitter streaming data running through the database instead of connecting to the simulated stream.

### 3.2 Experiments

For our experiments, we simulate a live stream of real-life Twitter data and analyze the streamed tweets. Since we only need the text of the tweets, we run a continuous filter query for the text field on the incoming stream and discard all other unused data fields. This helps us to minimize the amount of data to process during the analysis. The window size for this evaluation is set to one minute, which allows us to identify events within four minutes after the first triggering appearance of the corresponding event term. The minimum limit of the added up IDF shift value for a

<sup>2</sup><http://www.basex.org>

term to become an event term is set to 20% for single terms and 12% for bigrams. These values can be easily changed and dynamically adjusted. To track the evolution of identified events, we also take four minute windows to extract the top 10 most co-occurring terms of the event term. By manually evaluating the event terms and the most frequent co-occurrence terms of the event terms we can derive that the event terms are a mixture of non-english terms which are not filtered out by our analysis, names of famous people (e.g., *Justin Bieber*, *Chris Brown*), and real-world events.

The first experiment deals with the hours from 07:00 - 09:00 AM UTC on Wednesday, April 11th 2012. In this time frame our analysis identifies a total of 50 event terms for single terms and 21 event terms for bigrams within a total of 916,948 tweets. To further explain the usefulness of our approach, we choose the following two event terms. In Table 1, we can see how the event term “earthquake” in minute 8:45 with an overall shift of 22.13% evolved to an event term. Table 2 shows the evolution of the event term “tsunami” five minutes later in minute 8:50 with an overall shift of 27.67%. The evolution of the IDF value of the event terms in difference to the evolution of the IDF value of non-event terms can be seen in Figure 2. We can see that the IDF value of the event term “earthquake” increases significantly and therefore there is a high shift in the value. Furthermore, the event term “tsunami” shows almost the same behavior just five minutes later. In contrast to these terms, the two non-event terms “twitter” and “love” have almost no change in the IDF value over time.

	earthquake	average
<i>IDF Value Minute 8:42</i>	6.54	7.69
<i>IDF Value Minute 8:43</i>	6.70	8.09
<i>IDF Value Minute 8:44</i>	5.85	7.93
<i>IDF Value Minute 8:45</i>	5.15	7.85
<i>IDF Shift Minute 8:42-8:43</i>	-2.50%	-3.86%
<i>IDF Shift Minute 8:43-8:44</i>	12.72%	1.45%
<i>IDF Shift Minute 8:44-8:45</i>	11.91%	0.39%
<i>Total Shift</i>	22.13%	

Table 1: Detection of event term “earthquake” in minute 8:45.

	tsunami	average
<i>IDF Value Minute 8:47</i>	5.24	8.18
<i>IDF Value Minute 8:48</i>	4.76	8.29
<i>IDF Value Minute 8:49</i>	4.35	8.43
<i>IDF Value Minute 8:50</i>	3.92	8.39
<i>IDF Shift Minute 8:47-8:48</i>	9.24%	-0.99%
<i>IDF Shift Minute 8:48-8:49</i>	8.64%	-1.22%
<i>IDF Shift Minute 8:49-8:50</i>	9.79%	0.19%
<i>Total Shift</i>	27.67%	

Table 2: Detection of event term “tsunami” in minute 8:50.

The result of our event identification analysis indicates that an “earthquake” and a “tsunami” happened in the corresponding time frame. With the event tracking analysis we are able to extract more useful information about the identified events. After the event is detected, we analyze the new incoming tweets corresponding to that event and extract the ten most frequent co-occurrence terms for the time windows of four minutes. The ten most frequent common terms and their percentage frequency for the minutes after the event “earthquake” are the following:

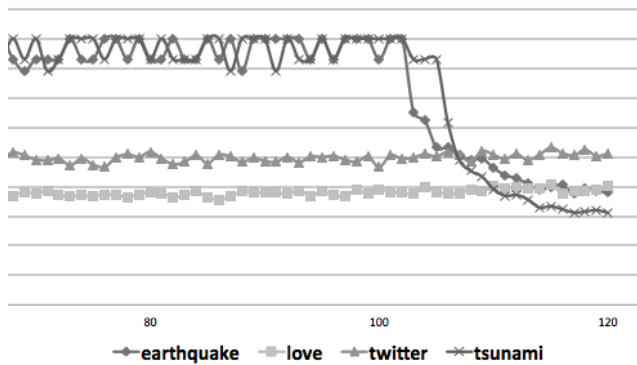


Figure 2: Sample IDF value evolution of the event terms “earthquake” and “tsunami” and the non-event terms “love” and “twitter” in the first experiment.

- *Minute 8:46 to Minute 8:49*: epicenter (7.39), aceh (6.82), banda (5.68), tsunami (5.11), chennai (5.11), office (4.55), depth (4.55), warning (4.55), malaysia (3.98), northern (3.41)
- *Minute 8:50 to Minute 8:54*: aceh (37.02), tsunami (25.39), warning (24.22), agency (23.26), issues (23.06), epicenter (17.83), sumatra (17.44), indonesia (17.25), coast (15.50), west (15.31)
- *Minute 8:55 to Minute 8:59*: aceh (42.13), tsunami (33.43), warning (30.62), agency (24.72), issues (24.72), indonesia (21.35), scale (17.42), magnitude (17.13), sumatra (16.85), richter (16.85)

We can see that in the first window the term “tsunami” is in only 5% of the tweets with “earthquake”. In the second window, however, there is a rapidly growing frequency of the term “tsunami”, which allows us to conclude that the topic drifts from discussions and news about the earthquake to messages about an expected or ongoing tsunami. Also further information like “aceh” (a city in Indonesia), “indonesia”, and “sumatra” is extracted by the event tracking phase. The extraction of the ten most frequent common terms identifies also terms like “aceh” or “warning”. With this information we are able to combine the two detected events (“earthquake” and “tsunami”) into one top event. The terms for the minutes after the “tsunami” event happened can be seen in the following:

- *Minute 8:51 to Minute 8:55*: gempa (64.96), peringatan (50.76), bengkulu (50.67), dini (50.49), lampung (50.49), sumut (50.13), sumbar (49.96), aceh (25.88), warning (13.84), earthquake (12.76)
- *Minute 8:56 to Minute 9:00*: gempa (52.53), peringatan (40.08), dini (39.45), bengkulu (39.03), lampung (38.71), sumbar (38.71), sumut (38.50), aceh (29.11), warning (18.78), earthquake (13.71)

The second experiment deals with the hours from 05:00 - 09:00 AM UTC on Friday, July 20th 2012. In this time frame our analysis identifies a total of 73 event terms for single terms and a total of 54 event terms for bigrams. Figure 3 shows the frequencies of the tweets and terms per minute, the overall total of all tweets for the four hours is 4,602,574

tweets. The frequency overview shows that we have an almost constantly number of terms and tweets in the windows over time. In Table 3, we can see how the event term “knight rises” in minute 7:20 with an overall shift of 14.21% evolved to an event term. Table 4 shows the evolution of the event term “aurora” almost an hour later in minute 8:19 with an overall shift of 20.64%.

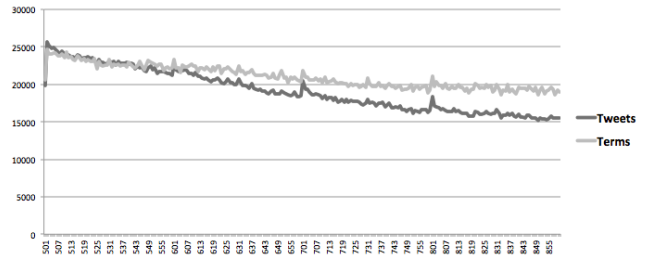


Figure 3: Frequency of the amount of tweets and terms per minute in the four hours of the second experiment.

	knight rises	average
<i>IDF Value Minute 5:51</i>	8.61	9.13
<i>IDF Value Minute 5:52</i>	7.81	9.13
<i>IDF Value Minute 5:53</i>	7.78	9.10
<i>IDF Value Minute 5:54</i>	7.59	9.11
<i>IDF Shift Minute 5:51-5:52</i>	9.32%	-0.19%
<i>IDF Shift Minute 5:52-5:53</i>	0.42%	0.10%
<i>IDF Shift Minute 5:53-5:54</i>	2.46%	-0.23%
<i>Total Shift</i>	12.20%	

Table 3: Detection of event term “knight rises” in minute 5:54.

	aurora	average
<i>IDF Value Minute 8:16</i>	8.59	8.84
<i>IDF Value Minute 8:17</i>	7.29	8.84
<i>IDF Value Minute 8:18</i>	6.96	8.82
<i>IDF Value Minute 8:19</i>	6.89	8.82
<i>IDF Shift Minute 8:16-8:17</i>	15.15%	0.00%
<i>IDF Shift Minute 8:17-8:18</i>	4.53%	0.02%
<i>IDF Shift Minute 8:18-8:19</i>	0.96%	-0.11%
<i>Total Shift</i>	20.64%	

Table 4: Detection of event term “aurora” in minute 8:19.

The result of our event identification analysis shows that a couple of events happened in the corresponding hour. In our case we are interested in two events. The first one is called “knight rises” and the second is “aurora”. Since we have no knowledge as to what these events are about, we use the results of the event tracking analysis to extract more useful information about the events. The ten most frequent co-occurrence terms and their percentage frequency for the minutes after the event “knight rises” are the following:

- *Minute 5:56 to Minute 6:00*: dark (96.43), movie (10.71), experience (7.14), century (7.14), line (7.14), cinemas (7.14), center (7.14), imax (7.14), theatre (7.14), river (3.57)

- *Minute 6:00 to Minute 6:04*: dark (100.00), cinemark (8.11), century (8.11), sinners (5.41), children (5.41), masked (5.41), imax (5.41), theaters (5.41), mamba (5.41), people (5.41)
- *Minute 6:04 to Minute 6:08*: dark (100.00), movie (9.52), people (7.14), batman (7.14), midnight (7.14), spiderman (4.76), regal (4.76), rumor (4.76), alert (4.76), spoiler (4.76)

By looking at the co-occurrence terms, we are able to figure out that most of the tweets discuss the “premiere” of a movie in an IMAX theatre at midnight. Since there is an ongoing premiere of the new movie “The Dark Knight Rises” on this day and time, we can conclude that the identified event is about the corresponding real-world event.

Since the analysis also detected an event “aurora” in minute 8:19, there could be a correlation between these two events. The extraction of the ten most frequent co-occurring terms for the event term “aurora” identifies the following terms:

- *Minute 8:20 to Minute 8:24*: shooting (78.69), colorado (75.41), film (67.21), premiere (67.21), people (65.57), dark (63.93), knight (63.93), local (62.30), media (60.66), rises (60.66), update (59.02)
- *Minute 8:24 to Minute 8:28*: colorado (70.45), shooting (61.36), dark (52.27), knight (52.27), film (45.45), premiere (45.45), people (43.18), rises (43.18), media (36.36), local (36.36), update (36.36)

We can derive that the newly detected event is somehow related to the earlier detected event “knight rises”. Since the first event describes the premiere of a new movie and the new event “aurora” describes a mass shooting happening during the movie premiere of “The Dark Knight Rises” in “Colorado”, we can conclude that there is a dependency between these two events.

## 4. RELATED WORK

The extreme popularity of Twitter and access to its public social data stream have resulted in an increasing amount of Twitter-related scientific, industrial, and governmental research initiatives. In this section, we summarize the most related work.

Bontcheva *et al.* [4] present an overview of sense making in social media data, which also includes current event detection methods in social media streams. They classified detection methods into three categories: clustering-based, model-based, and those based on signal processing.

An event detection system dedicated to earthquakes is presented by Sakaki *et al.* [14]. In contrast to our approach, they use the keyword search feature provided by the Twitter API to gather data in specified time intervals. Schühmacher *et al.* [15] propose another domain-specific event detection method on microblogs to support forensic analysis. They train a linear classifier to detect suspicious posts. Weng *et al.* [18] use wavelet analysis on frequency-based raw signals of terms from tweets for detecting events. They use a keyword-filtered dataset to show their practical usage for identifying events during the Singapore General Election in 2011. Marcus *et al.* [10] demonstrate an application called “TwitInfo”, which identifies and labels event peaks for given search queries related to the event. In contrast to our proposed idea, which uses an unfiltered data stream, all of the above mentioned systems are somehow restricted.

More recently, Ritter *et al.* [13] presented the first approach for open domain event extraction from Twitter. Their approach is based on latent variable models and proceeds by first discovering event types, which match the data and then using these results to classify aggregate events. However, no discussion about applying this approach directly to the streaming data is included. Alvanaki *et al.* [3] proposed a system “enBlogue”, which analyzes statistics about tags and tag pairs for identifying unusual shifts in correlations. Further recent work proposed by Nishida *et al.* [12] shows a classification model of tweet streams for identifying changes in statistical properties on word basis, which is used for topic classification.

General research on on-line event detection has a long track record. In 1998, Yang *et al.* [19] published a study about retrospective and on-line event detection. They used text retrieval and clustering techniques for detecting events in a temporally ordered stream of news stories. In the same year, Allan *et al.* [2] focused on a strict on-line setting by using a modified single-pass clustering approach for event detection and information filtering for event tracking. However these two approaches used clean and well-formed news stories as sources for detecting events.

## 5. CONCLUSIONS

In this paper, we presented a method for identifying events in the real-world social media data streams of Twitter. We have shown that by means of aggregation it is possible to handle large volumes of data and gain important insights into it. We believe that under ideal conditions the data streamed by Twitter can support faster detection of events than by using reports of news agencies. Although we obtained the data through the Twitter API that only provides 10% of the total data stream, which might introduce a skew in the tweets we analyze, the total stream can be assumed to contain more complete information an event.

Our evaluation shows that we are able to identify events as well as to track the progress of the event and the context around it in a simulated environment. However, the identification also detects a certain amount of non-event terms as events. This is an indication that the identification phase needs to be improved by including more information, like geographical data or other features extracted from the metadata. The tracking of the events shows that the context around an event can be described properly and it is also possible to identify relationships and dependencies between events. For example, in both experiments we were able to draw the conclusion that the second identified event is a follow-up or related event of the first one. With the continuous removal of event candidates from the term set, we are able to scale to the amount and the speed of tweets and terms in the streaming data.

## 6. FUTURE WORK

A first extension of our approach will be the integration of further information into the event identification phase. This goal can be achieved by using information from the metadata fields or by extracting more information from the textual content of the tweets. In addition to the actual content of the tweet messages, Twitter provides 60 metadata fields describing the tweet (e.g., count of retweets, geographic location) and the user’s profile (e.g., count of followers). This

additional knowledge can be used to extract further characteristics of the identified events. For example, if a majority of tweets related to an event have similar geographical information (such as the same city or country), one can assume that the event possibly originated at that location. Furthermore, it will be an interesting task to implement a categorization and ranking (e.g., globally important) analysis for the detected events. To support the ranking of events, we can also integrate the metadata (e.g., number of retweets vs. number of independent tweets) in our analysis.

A further extension is the integration of additional data sources. Stock exchange markets, weather forecasts, data from news agencies, RSS feeds, and further social media services offer contents that can be retrieved in different ways as streams and could also enrich our event identification and tracking analysis. For example, even the social media photo sharing platform Flickr was recently used as data source for event detection [6].

For evaluation purposes it would be interesting to evaluate our approach against more complex state of the art approaches, such as the one presented by Weng *et al.* [18]. This line of future work would enable us to better understand how much complexity is needed to differentiate between event terms and standard terms.

## 7. REFERENCES

- [1] I. Adã and M. R. Berthold. Unifying Change – Towards a Framework for Detecting the Unexpected. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining Workshops, ICDMW '11*, pages 555–559, Washington, DC, USA, 2011. IEEE Computer Society.
- [2] J. Allan, R. Papka, and V. Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '98*, pages 37–45. ACM, 1998.
- [3] F. Alvanaki, S. Michel, K. Ramamritham, and G. Weikum. See what's enblogue: real-time emergent topic identification in social media. In *Proceedings of the 15th International Conference on Extending Database Technology, EDBT '12*, pages 336–347, New York, NY, USA, 2012. ACM.
- [4] K. Bontcheva and D. Rout. Making Sense of Social Media Streams through Semantics: a Survey. *Semantic Web*, 2012.
- [5] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), 2009.
- [6] L. Chen and A. Roy. Event detection from Flickr data through wavelet-based spatial analysis. In *Proceedings of the 2009 ACM CIKM International Conference on Information and Knowledge Management (CIKM '09)*, 2009.
- [7] A. Dries and U. Rückert. Adaptive concept drift detection. *Stat. Anal. Data Min.*, 2:311–327, 2009.
- [8] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. In *ACL (Short Papers)*, pages 42–47, 2011.
- [9] D. Kifer, S. Ben-David, and J. Gehrke. Detecting change in data streams. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30, VLDB '04*, pages 180–191. VLDB Endowment, 2004.
- [10] A. Marcus, M. S. Bernstein, O. Badar, D. R. Karger, S. Madden, and R. C. Miller. Twitinfo: aggregating and visualizing microblogs for event exploration. In *Proceedings of the 2011 annual conference on Human factors in computing systems, CHI '11*, pages 227–236. ACM, 2011.
- [11] M. Markou and S. Singh. Novelty detection: A review - part 1: Statistical approaches. *Signal Processing*, 83:2003, 2003.
- [12] K. Nishida, T. Hoshida, and K. Fujimura. Improving tweet stream classification by detecting changes in word probability. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval, SIGIR '12*, pages 971–980, New York, NY, USA, 2012. ACM.
- [13] A. Ritter, Mausam, O. Etzioni, and S. Clark. Open domain event extraction from twitter. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '12*, pages 1104–1112, New York, NY, USA, 2012. ACM.
- [14] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes Twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web, WWW '10*, pages 851–860. ACM, 2010.
- [15] J. Schühmacher and C. Koster. Signalling events in text streams. In P. Daras and O. Mayora-Ibarra, editors, *UCMedia*, volume 40 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 335–339. Springer, 2009.
- [16] K. Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval*, pages 132–142. Taylor Graham Publishing, 1988.
- [17] Twitter Team. Developing for @twitterapi (Techcrunch Disrupt Hackathon), 2012, <https://dev.twitter.com/docs/intro-twitterapi>.
- [18] J. Weng, Y. Yao, E. Leonardi, and F. Lee. Event Detection in Twitter. Technical report, HP Labs, 2011.
- [19] Y. Yang, T. Pierce, and J. Carbonell. A study of retrospective and on-line event detection. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '98*, pages 28–36. ACM, 1998.

# Recommendation of Multimedia Objects for Social Network Applications

Flora Amato, Francesco Gargiulo, Vincenzo Moscato, Fabio Persia, Antonio Picariello  
University of Naples Federico II  
{flora.amato,francesco.gargiulo,vmoscato,fabio.persia,picus}@unina.it

## ABSTRACT

Recommender systems help people in retrieving information that match their preferences by recommending products or services from a large number of candidates, and support people in making decisions in various contexts: what items to buy, which movie to watch or even who they can invite to their social network. They are especially useful in environments characterized by a vast amount of information, since they can effectively select a small subset of items that appear to fit the user's needs.

We present the main points related to recommender systems using multimedia data, especially for social networks applications. We also describe, as an example, a framework developed at the University of Naples "Federico II". It provides customized recommendations by originally combining intrinsic features of multimedia objects (low-level and semantic similarity), past behavior of individual users and overall behavior of the entire community of users, and eventually considering users' preferences and social interests.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous;  
H.5.1 [Information Interfaces And Presentation]: Multimedia Information Systems

## General Terms

Multimedia Recommender System, Social Networks

## 1. INTRODUCTION

Images, Audios and Videos are the kind of data most involved in Social Networks applications, due to the extraordinary technological progress that makes possible the generation and exchange of multimedia content at low cost and in a very easy way: just to make an example, tons of short messages, images and video are produced and exchanged using smart phone, pads and laptop, and are posted every day over popular social networks (e.g. Facebook, Twitter etc.).

As a consequence, massive collections of multimedia objects are now widely available to a large population of users and to the appealing and sophisticated tools and applications used for social networks. However, the retrieval of such objects and, in particular, of the "right" multimedia component that can be suitable for a certain application, still remains a challenging problem. To this aim, a number of algorithms and tools – generally referred to as *Recommender Systems* – are being proposed to facilitate browsing of these large data repositories.

We note that the use of recommender systems is one of the steps for realizing the transition from the *era of search* to the *era of discovery*: that is, according to *Fortune* magazine writer Jeffrey M. O'Brien, *search is what you do when you are looking for something while discovery is when something wonderful that you didn't know existed, or didn't know how to ask for, finds you*.

Recommender systems help people in retrieving information that match their preferences by recommending products or services from a large number of candidates, and support people in making decisions in various contexts: what items to buy [1], which movie to watch [2] or even who they can invite to their social network [3]. They are especially useful in environments characterized by a vast amount of information, helping people to effectively select a small subset of items that fit the user's needs [4, 5]. This kind of systems can be used in many contexts, as example in Cultural Heritage, for guiding the tourists in their browsing activities or, in e-Commerce, for suggesting items that the users are likely to buy.

Here, we present the main points related to recommender systems using multimedia data, especially for social networks applications. We also describe, as an example, a framework developed at the University of Naples "Federico II". It provides customized recommendations by originally combining intrinsic features of multimedia objects (low-level and semantic similarity), past behavior of individual users and overall behavior of the entire community of users, and eventually considering users' preferences and social interests.

## 2. THEORETICAL BACKGROUND

Wikipedia defines Recommender Systems as "a specific type of information filtering technique that attempts to present information items (movies, music, books, news, images, web pages, etc.) that are likely of interest to the user".

However, this definition shows only one basic idea of the number of features and aspects of modern recommender systems.

From a general point of view, recommenders are a class of applications that try to predict users choices [6]. Just to bring the problem into focus, some good examples of recommendation systems are described in the following.

*Product Recommendations* - perhaps the most important use of recommendation systems; on-line vendors present each returning user with some suggestions of products that they might like to buy on the base of the purchasing decisions made by similar customers. *Movie Recommendations* - in MovieLens.org, for example, users initially rate some subset of movies that they have already seen, with ratings specified on a scale from 1 to 6 stars, where 1 is “Awful” and 6 is “Must See”; MovieLens then uses the ratings of the community to recommend other movies that user might be interested in or to predict how that user might rate a movie; therefore, the recommendation engine should be able to estimate (predict) the ratings of non-rated movie/user combinations and generate appropriate recommendations based on these predictions. *News Articles* - news services have attempted to identify articles of interest to readers, based on the articles that they have read in the past. *Social Recommendations* - recommending new friends in social nets, such as Facebook, Twitter, and so on based on common interests retrieved from the analysis of the published posts.

In these examples we recognize different kinds of recommendation, such as *personalized recommendation* - based on the individual’s past behavior, *social recommendation* - based on the past behavior of similar users, *item recommendation* - based on features related to the item itself and any *combination* of the approaches above.

What is an “item”? In real applications it is more than simple text: it includes images, audio and video streams, and it is not ensured the presence of and adequate metadata system. In this framework, the challenge is then how to exploit the information given by multimedia objects, and how to combine it with past behavior of the single user or of a community of users in order to provide easy and effective recommendations.

Let us give some formal description of the above described problem.

A recommendation system deals with a set of users  $U = \{u_1, u_2, \dots, u_i, \dots, u_m\}$  and a set of objects  $O = \{o_1, o_2, \dots, o_j, \dots, o_n\}$ <sup>1</sup>. For each pair  $(u_i, o_j)$ , a recommender can compute a score  $r_{i,j}$  that measures the expected interest of user  $u_i$  in object  $o_j$  (or the expected utility of object  $o_j$  for user  $u_i$ ), using a knowledge base and a *scoring* (or *ranking*) algorithm that should take into account that users preferences change with context. In other terms, for each user  $u \in U$ , the recommendation problem is to choose a set of items in  $O$  that maximize the user’s utility, given the current context.

In this formulation, the utility of an item is usually represented by a rating which can be an arbitrary function, including a profit function.

Depending on the application, utility can either be specified by the user, as it is often done for user-defined ratings, or computed by the application, as in profit-based utility functions. Each user in  $U$  can be associated with a profile that includes various characteristics, such as age, gender, income, marital status, and so on; similarly, each data item in  $O$  is associated with a set of features.

<sup>1</sup>Both  $O$  and  $U$  can be very large, in the order of thousands or even millions of items, such as in book recommendations systems

For instance, in a movie recommendation application,  $O$  being a collection of movies, each movie can be represented by its title, genre, director, year of release, main actors, etc.

The utility is usually not defined on the whole  $U \times O$  space, but only on some subset of it, and thus the central problem is to extrapolate  $r$  to the whole space  $U \times O$ .

Extrapolations from known to unknown ratings are usually done by: (i) specifying heuristics that define the utility function and empirically validating its performances and (ii) estimating the utility function that optimizes certain performance criterion, such as the mean square error. Once the unknown ratings are estimated, actual recommendations of an item to a user are made by selecting the highest rating among all the estimated ratings for that user. Alternatively, we can recommend the  $N$  best items to a user or a set of users to an item.

As an example of novel and effective recommendation techniques, in the following we show a system for recommendations of multimedia items, that originally combines intrinsic features of multimedia objects (low-level and semantic similarity), past behavior of individual users, and overall behavior of the entire community of users and eventually considering users’ preferences and social interests. Recommendations are ranked using an importance ranking algorithm that resembles the well known *PageRank* [7] ranking strategy.

In order to best understand the described system, let us consider a typical scenario, where an effective multimedia recommender system would be desirable in social networks applications. In particular, let us consider popular social networks (e.g. Facebook, Twitter, Flickr) that, supporting an intelligent browsing of images’ collection, allow users to quickly retrieve her or of her friends’ pictures with respect to a given category (e.g. landscapes, animals, vacation, etc.) in order to automatically create personalized photographic album. For example, if the user wants to create an album of *London* using photos of her last vacation and other images of her friends that have just visited the city, an image recommender systems should be able to suggest all the similar images with respect to that observed by the user considering: similarities among images, past behaviors of the users community, users social interests and preferences.

### 3. RELATED WORK

Literature about recommender systems is reported in different surveys[4, 8, 9], in which these systems are broadly classified into three major categories: *content based recommendation*, *collaborative filtering* and *hybrid approaches* (that combine collaborative and content-based methods), using the main results in cognitive sciences, approximation theory, information retrieval, and forecasting theories.

Recommendations based on content-based approach arise from research in information retrieval and information filtering [10]: in such frameworks, the suggested item is in some way “similar” to the previous ones; for example, if a user would like to have a recommendation about a good movie to watch, the system tries to understand the commonalities among the movies the user  $u$  has rated highly in the past (specific actors, directors, genres, subject matter, etc), and only the movies that have a high degree of similarity to the user’s preferences would be recommended. In recent times, the content-based approach has been applied to other types of multimedia data as in [11], [12].

Collaborative filtering techniques associate each user to a set of other users having *similar* profiles and recommending items based on the similarity between users, rather than on the similarity between data items themselves [13, 14]. In other terms, this process uses the opinions of other people, shares opinions with others, as in real life peoples discusses about fashion or places to visit [4], [9].

The combination of collaborative and content based methods is at the basis of hybrid approaches; as reported in [4], they are usually classified into different techniques: (i) implementation of collaborative and content-based methods separately and combination their predictions [15], (ii) incorporation of some content-based characteristics into a collaborative approach [16, 17], (iii) incorporation of some collaborative characteristics into a content-based approach [18], and (iv) construction of a general unifying model that incorporates both content-based and collaborative characteristics [19].

Hybrid recommendation systems can also use knowledge-based techniques, such as case-based reasoning, in order to improve recommendation accuracy and to address some of the limitations (e.g., new user, new item problems) of traditional recommender systems [20, 21]. Some authors [16, 17, 18] empirically compare the performance of the hybrid with the single ones and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches [22].

In the last years approaches based on the use of knowledge representation techniques and semantic relationships among recommended objects have been proposed. For example, in [23] the authors propose an approach that combines the on-line user's personal preferences, general user's common preference from users' most recent experiences, and experts knowledge for personalized recommendations.

Moreover, a recommending approach that shows ranked news to users, considering previous visits, the terms contained in articles and the category they are assigned to is presented in [24] and [25]. The authors designed two probabilistic models based on the aspect model to identify semantic relationships [26] in user access to classified news.

#### 4. THE RECOMMENDATION STRATEGY

So far we have discussed that a multimedia recommender system for multimedia collections in a social network context has to provide the capability of reliably identifying those objects that are most likely to match the interests of a user at any given point of her exploration.

Generally, we have to address four fundamental questions: (i) How can we select a set of objects from the collection that are good candidates for recommendation? (ii) How can we rank the set of candidates? (iii) How can we capture, represent and manage semantics related to multimedia objects to reduce the semantic gap between what user is watching and what she is looking for? (iv) How can we arrange the recommended objects considering users' preferences and social interests?

To give an answer to the first two questions, we adopt a recommendation strategy based on an importance ranking method that strongly resembles the *PageRank* ranking system and that the authors proposed in [7, 27].

Our basic idea is to assume that when an object  $o_i$  is chosen after an object  $o_j$  during the same browsing session, this event means that  $o_j$  "is voting" for  $o_i$ .

Similarly, the fact that an object  $o_i$  is very similar to  $o_j$  can also be interpreted as  $o_j$  "recommending"  $o_i$  (and vice-versa). Thus, we model a browsing system for a set of objects  $O$  as a labeled graph  $(G, l)$ , where  $G=(O, E)$  is a directed graph and  $l: E \rightarrow \{pattern, sim\} \times R^+$  is a function that associates each edge in  $E \subseteq O \times O$  with a pair  $(t, w)$ , where  $t$  is the type of the edge which can assume two enumerative values (*pattern* and *similarity*) and  $w$  is the weight of the edge. According to this model, we list two different cases: (i) a *pattern label* for an edge  $(o_j, o_i)$  denotes the fact that an object  $o_i$  was accessed immediately after an object  $o_j$  and, in this case, the weight  $w_{ij}$  is the number of times  $o_i$  was accessed immediately after  $o_j$ ; (ii) a *similarity label* for an edge  $(o_j, o_i)$  denotes the fact that an object  $o_i$  is similar to  $o_j$  and, in this case, the weight  $w_{ij}$  is the similarity between  $o_j$  and  $o_i$ . In other terms, a link from  $o_j$  to  $o_i$  indicates that part of the importance of  $o_j$  is transferred to  $o_i$ .

Given a labeled graph  $(G, l)$ , we can formulate the definition of *preference grade* of an object  $o_i$  as follows:

$$\rho(o_i) = \sum_{o_j \in P_G(o_i)} w_{ij} \cdot \rho(o_j) \quad (1)$$

where  $P_G(o_i) = \{o_j \in O | (o_j, o_i) \in E\}$  is the set of predecessors of  $o_i$  in  $G$ , and  $w_{ij}$  is the normalized weight of the edge from  $o_j$  to  $o_i$ . For each  $o_j \in O$   $\sum_{o_i \in S_G(o_j)} w_{ij} = 1$  must hold, where  $S_G(o_j) = \{o_i \in O | (o_j, o_i) \in E\}$  is the set of successors of  $o_j$  in  $G$ .

It is easy to see that the vector  $R = [\rho(o_1) \dots \rho(o_n)]^T$  can be computed as the solution to the equation  $R = C \cdot R$ , where  $C = \{w_{ij}\}$  is an ad-hoc matrix that defines how the importance of each object is transferred to other objects and can be seen as a linear combination of the following elements [7].

A *local browsing matrix*  $A_l = \{a_{ij}^l\}$  for each user  $u_i \in U$ . Its generic element  $a_{ij}^l$  is defined as the ratio of the number of times object  $o_i$  has been accessed by user  $u_i$  immediately after  $o_j$  to the number of times any object in  $O$  has been accessed by  $u_i$  immediately after  $o_j$ . A *global browsing matrix*  $A = \{a_{ij}\}$ . Its generic element  $a_{ij}$  is defined as the ratio of the number of times object  $o_i$  has been accessed by any user immediately after  $o_j$  to the number of times any object in  $O$  has been accessed immediately after  $o_j$ . A *multimedia similarity matrix*  $B = \{b_{ij}\}$  such that  $b_{ij} = \sigma(o_i, o_j) / \Gamma$  if  $\sigma(o_i, o_j) \geq \tau \forall i \neq j$ , 0 otherwise.  $\sigma$  is any similarity function defined over  $O$  which calculates for each couple of objects their multimedia relatedness in terms of low (features) and high level (semantics) descriptors;  $\tau$  is a threshold and  $\Gamma$  is a normalization factors which guarantees that  $\sum_i b_{ij} = 1$ .

Matrix  $B$  allows to address the third question that we introduced at the beginning of the section and thus to introduce a sort of content-based retrieval in the recommendation process.

In particular, to compute  $B$  matrix in the image realm, we can adopt the most diffused multimedia features (Tamura descriptors, MPEG-7 color-based descriptors, MPEG-7 edge-based descriptors, MPEG-7 color layout- based descriptors and all MPEG7 descriptors) and the related similarity metrics. In addition, we can exploit specific image metadata – depending on the considered domain – and the semantic similarity can be computed using the most diffused metrics for semantic relatedness of concepts based on a vocabulary (Li-Bandar-McLean, Wu-Palmer, Rada, Leacock-Chodorow, Budanitsky).



As an example, in [27], we consider the set of digital paintings belonging to a social network related to Cultural Institutions and the semantic similarity combines similarities among *artists*, *genres* and *subjects* metadata obtained by using a fixed taxonomy produced by domain experts with image features. The combination between high and low level descriptors is based on Sugeno fuzzy integral of Li and MPEG-7 color layout- based similarities, and Sugeno fuzzy integral of Wu-Palmer and MPEG-7 color based similarities, in order to have more high level values of precision and recall; thus we used this last combination for matrix  $B$  computation.

Still remains to discuss how to compute customized rankings for each individual user considering user context information. In this case, we can then rewrite previous equation considering the ranking for each user as  $R_i = C \cdot R_i$ , where  $R_i = [\rho(o_i) \dots \rho(o_n)]^T$  is the vector of preference grades, customized for a user  $u_i$ .

We note that solving equation  $R = C \cdot R$  corresponds to find the stationary vector of  $C$ , i.e., the eigenvector with eigenvalue equal to 1. In [7], it has been demonstrated that  $C$ , under certain assumptions and transformations, is a real square matrix having positive elements, with a unique largest real eigenvalue and the corresponding eigenvector has strictly positive components. In such conditions, the equation can be solved using the *Power Method* algorithm.

It is important to note that  $C$  takes into account the user's context and does not have to be computed for all the database objects, but it needs to be computed only for those objects that are good candidates, i.e. the most *similar* objects to that a user is currently watching (*pre-filtering strategy*).

Finally, to met the last question, the set of suggested items is organized in apposite recommendation lists: they are not fixed and are arranged on the base of *social* user interests and preferences in terms of *taxonomic attributes* – e.g. favorite artists, genres and subjects –, which values can either retrieved using proper questionnaires or gathered by means of apposite API from the most diffused social networks. The preference degree of objects, which do not reflect user needs in terms of semantic similarities, are penalized and such objects could be excluded from recommendation (*post-filtering strategy*).

## 5. SYSTEM DESCRIPTION

Figure 1 shows an overview of our multimedia recommender system, which takes as input the current context in terms of observed objects (e.g. an image) and generates a list of items. We distinguish the following components.

*Items Manager* - It is a repository manager that stores the items to be suggested with the related descriptions. In the case of images, it consists of an image DBMS, storing raw data with the related low level features and metadata. *Users Log Tracker* - It is a module devoted to capture and store - in an appropriate format - all the users' browsing sessions in terms of accessed items during their explorations. *User Preferences Manager* - It is a module devoted to gather from social networks and manage all the user social interests and preferences in terms of taxonomic attributes values. *Items Deliverer* - It aims at delivering recommended objects to each user in a format that will depend on the user profile and device. *Recommendation Engine* - It is the system core that for each user and on the base of current context dynam-

ically proposes a set of recommended objects ordered on the base of their utility. In particular, it is composed by: (i) a *Browsing Matrices Computation* Module - able to transform the collected browsing sessions into two matrices: a global matrix which takes into account the overall browsing behavior of the users, and a local matrix which considers the behavior of a single user; (ii) a *Similarity Matrix Computation* Module - capable of computing a similarity degree for each couple of objects and storing such degrees into a matrix; (iii) a *Candidate Set Building* Module - computes the subset of items that are more suitable for users needs; (iv) a *Items Ranks Computation* Module - performs the ranking and post-filtering of the selected candidates for recommendation.

As discussed in [27], the system performances in terms of user's satisfaction are encouraging, providing a better (less frustrating) user experience during assigned browsing tasks with respect to classical image retrieval systems.

### 5.1 Application examples in Social Networks

The system is a platform that can provide services for many social network applications. Just to make few examples, in the case of image collection, we use recommendation services to assist users during browsing of image gallery containing objects with the same subject (e.g. landscape, animal) or to suggest the most effective tags for image indexing or to automatically create personalized photographic album. For audio and video data, we can exploit recommendation services to create personalized play-lists using, for example, *Youtube* linked data.

## 6. PRELIMINARY EXPERIMENTS

Recommender systems are generally complex applications that are based on a combination of several models, algorithms and heuristics. Recently, researchers began examining issues related to users' subjective opinions and developing additional criteria to evaluate recommender systems. In particular, they suggest that user's satisfaction does not always (or, at least, not only) correlate with the overall recommender's accuracy and evaluation frameworks for measuring the perceived qualities of a recommender and for predicting user's behavioral intentions as a result of these qualities should be taken into account.

Starting from these considerations and based on current trends in the literature, in [28] we decided to perform both a user-centric evaluation and a more traditional evaluation based on well-established accuracy metrics.

In particular, the proposed evaluation strategy aimed at measuring (i) user satisfaction with respect to assigned browsing tasks, and (ii) effectiveness of the system in terms of accuracy.

In this work, we in turn evaluated the improvement of accuracy performances due to the post-filtering strategy based on users' social preferences. We used the dataset provided by the <http://www.grouplens.org> website, which makes available data collected by the *MovieLens* recommender system. Through its website, *MovieLens* collects the preferences expressed by a community of registered users on a huge set of movie titles. The dataset contains (i) explicit ratings about 1682 movies made by 943 users (only users who have rated at least 20 movies are considered), (ii) demographic information about users (age, gender, occupation, zip code), and (iii) a brief description of the movies (title, year, genres).

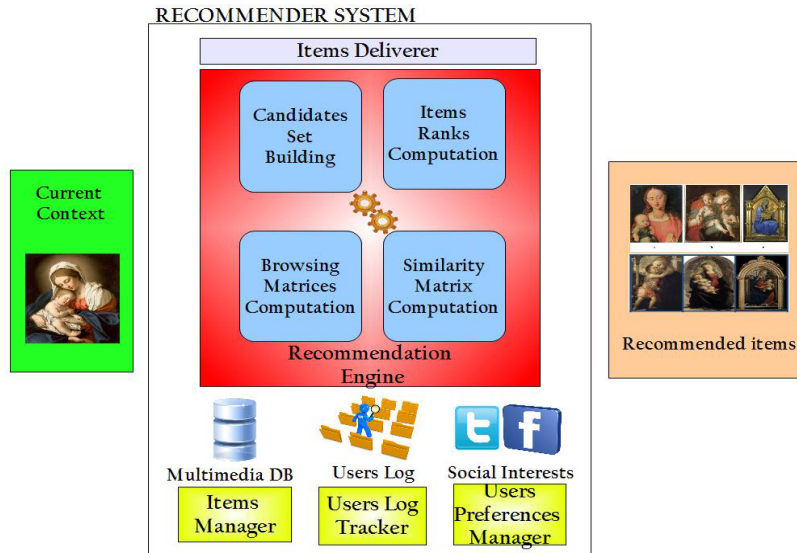


Figure 1: System Overview

Sparsity	Strategy	$RMSE$
0.7	Without post-filtering	0.95
0.7	Users' preferences post-filtering	0.87
0.75	Without post-filtering	0.95
0.75	Users' preferences post-filtering	0.95
0.8	Without post-filtering	1.02
0.8	Users' preferences post-filtering	0.97
0.85	Without post-filtering	1.07
0.85	Users' preferences post-filtering	0.99
0.90	Without post-filtering	1.15
0.90	Users' preferences post-filtering	1.04
0.95	Without post-filtering	1.32
0.95	Users' preferences post-filtering	1.16

Table 1: Accuracy Improvement using the post-filtering strategy

The experiments have been conducted on a collection of about 1,000 movies, rated by a subset of 100 users: each of them had rated at least 150 movies and at most 300, assigning each movie a score between 1 (“Awful”) and 5 (“Must to see”). Additionally, using the timestamp information, we were able to reconstruct usage patterns for each user and consequently the browsing matrices.

We compared in Table 1 the accuracy in terms of Root Mean Square Error of the predictions computed by our recommender system without and with the post-filtering strategy. In particular, we selected 50 test users (whose social preferences are captured by proper questionnaires) and computed the average accuracy for 50 predictions on a subset of the most recently observed items, increasing data sparsity for the same users.

## 7. FUTURE WORK

Recommender systems made a significant progress over

the last decade when numerous methods and several systems have been proposed. However, despite all these advances, the current generation of recommender systems still requires further improvements. These extensions include the improved modeling of users and multimedia large items' collections, generally depending on the considered application, incorporation of the contextual information into the recommendation process, support for multi-criteria ratings, and provision of a more flexible and less intrusive recommendation process [4].

Our proposal represents an extension of a hybrid recommender system supporting intelligent browsing of multimedia collections in social networks domain. We have shown that the customized recommendations may be computed combining several features of multimedia objects, past behavior of individual users and overall behavior of the entire community of users and users' social preferences. These techniques, described into details for images, can be easily adapted and extended to several kinds of multimedia data such as video, audio and texts.

In according to the research future directions, the system could be improved: (i) introducing explicit user profiling mechanism based on the creation of users categories, (ii) scaling the systems for large multimedia data collections, (iii) integrating the several strategies using SOAP as a built-in service for popular social networks.

## 8. REFERENCES

- [1] Xinrui Zhang and Hengshan Wang. Study on recommender systems for business-to-business electronic commerce. *Communications of the IIMA*, 5:53–61, 2005.
- [2] Song Qin, Ronaldo Menezes, and Marius Silaghi. A recommender system for youtube based on its network of reviewers. In Ahmed K. Elmagarmid and Divyakant Agrawal, editors, *SocialCom/PASSAT*, pages 323–328. IEEE Computer Society, 2010.

- [3] Przemyslaw Kazienko and Katarzyna Musial. Recommendation framework for online social networks. In Mark Last, Piotr S. Szczepaniak, Zeev Volkovich, and Abraham Kandel, editors, *Advances in Web Intelligence and Data Mining*, volume 23 of *Studies in Computational Intelligence*, pages 111–120. Springer, 2006.
- [4] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17:734–749, 2005.
- [5] Saverio Perugini, Marcos André Gonçalves, and Edward A. Fox. Recommender systems research: A connection-centric survey. *J. Intell. Inf. Syst.*, 23:107–143, September 2004.
- [6] A.G. Parameswaran, H. Garcia-Molina, and J.D. Ullman. Evaluating, combining and generalizing recommendations with prerequisites. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 919–928. ACM, 2010.
- [7] Massimiliano Albanese, Antonio d’Acierno, Vincenzo Moscato, Fabio Persia, and Antonio Picariello. Modeling recommendation as a social choice problem. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys ’10, pages 329–332, New York, NY, USA, 2010. ACM.
- [8] Michael J. Pazzani and Daniel Billsus. *The Adaptive Web: Methods and Strategies of Web Personalization*, volume 4321 of *Lecture Notes in Computer Science*, chapter Content-Based Recommendation Systems, pages 325–342. Springer, 2007.
- [9] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.
- [10] Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin. *COMMUNICATIONS OF THE ACM*, 35(12):29–38, 1992.
- [11] Veronica Maidel, Peretz Shoval, Bracha Shapira, and Meirav Taieb-Maimon. Evaluation of an ontology-content based filtering method for a personalized newspaper. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys ’08, pages 91–98, New York, NY, USA, 2008. ACM.
- [12] Katarzyna Musial, Krzysztof Juszczyszyn, and Przemyslaw Kazienko. Ontology-based recommendation in multimedia sharing systems. *System Science*, 34:97–106, 2008.
- [13] Jae Kyeong Kim, Hyea Kyeong Kim, and Yoon Ho Cho. A user-oriented contents recommendation system in peer-to-peer architecture. *Expert Syst. Appl.*, 34(1):300–312, 2008.
- [14] Hyea Kyeong Kim, Jae Kyeong Kim, and Young U. Ryu. Personalized recommendation over a customer network for ubiquitous shopping. *IEEE Trans. Serv. Comput.*, 2(2):140–151, 2009.
- [15] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*, 1999.
- [16] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72, March 1997.
- [17] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the Eighteenth national conference on Artificial intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [18] Ian Soboroff. Charles Nicholas and Charles K. Nicholas. Combining content and collaboration in text filtering. In *In Proceedings of the IJCAI’99 Workshop on Machine Learning for Information Filtering*, pages 86–91, 1999.
- [19] Andrew I. Schein, Alexandrin Popescul, Lyle H. Ungar, and David M. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the ACM SIGIR ’02*, pages 253–260, New York, NY, USA, 2002. ACM.
- [20] Robin Burke. Knowledge-based Recommender Systems. In *Encyclopedia of Library and Information Systems*, volume 69, 2000.
- [21] Stuart E. Middleton, Nigel R. Shadbolt, and David C. De Roure. Ontological user profiling in recommender systems. *ACM Trans. Inf. Syst.*, 22:54–88, January 2004.
- [22] Flora Amato, Antonino Mazzeo, Vincenzo Moscato, and Antonio Picariello. Building and retrieval of 3d objects in cultural heritage domain. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on*, pages 816–821. IEEE, 2012.
- [23] Chunyan Miao, Qiang Yang, Haijing Fang, and Angela Goh. A cognitive approach for agent-based personalized recommendation. *Know.-Based Syst.*, 20(4):397–405, 2007.
- [24] Sergio Cleger-Tamayo, Juan M. Fernández-Luna, and Juan F. Huete. Top-n news recommendations in digital newspapers. *Know.-Based Syst.*, 27:180–189, 2012.
- [25] Flora Amato, Antonino Mazzeo, Vincenzo Moscato, and Antonio Picariello. Exploiting cloud technologies and context information for recommending touristic paths. In *Intelligent Distributed Computing VII*, pages 281–287. Springer, 2014.
- [26] Flora Amato, Antonino Mazzeo, Vincenzo Moscato, and Antonio Picariello. Semantic management of multimedia documents for e-government activity. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS’09. International Conference on*, pages 1193–1198. IEEE, 2009.
- [27] Massimiliano Albanese, Antonio d’Acierno, Vincenzo Moscato, Fabio Persia, and Antonio Picariello. A multimedia recommender system. *ACM Transactions on Internet Technology (TOIT)*, 13(1):3, 2013.
- [28] Massimiliano Albanese, Antonio d’Acierno, Vincenzo Moscato, Fabio Persia, and Antonio Picariello. A multimedia recommender system. *ACM Trans. Internet Techn.*, 13(1):3, 2013.

# Estimating Completeness in Streaming Graphs

Malay Bhattacharyya  
Department of C.S.E.  
University of Kalyani  
malaybhattacharyya  
@klyuniv.ac.in

Supratim Bhattacharya  
Department of C.S.E.  
University of Kalyani  
bhattacharya.supratim  
@gmail.com

Sanghamitra  
Bandyopadhyay\*  
Machine Intelligence Unit  
Indian Statistical Institute  
sanghami@isical.ac.in

## ABSTRACT

Finding the completeness of a graph is important from various aspects. Considering the massive growth and dynamics of real-life networks, we readdress this problem in a streaming setting. We approach the problem of verifying the completeness of a graph by estimating the eigen values of a sketch of its adjacency matrix. Here, we provide the first approximation algorithm for estimating the completeness of a bipartite graph in the streaming model. The approach is further generalized for any arbitrary simple graph. We employ some useful recent results on  $\ell_1$  heavy eigen-hitters to construct the algorithms working in linear time and consuming sublinear space. The implementation of the algorithms have also been done and tested on a couple of networks. We illustrate the effectiveness of the proposed approaches in analyzing social, biological and other real-life networks.

## Categories and Subject Descriptors

E.1 [Data Structures]: Graphs and Networks; F.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems; G.2.2 [Discrete Mathematics]: Graph Theory

## General Terms

Theory, Design, Analysis

## Keywords

Streaming model, complete graphs, heavy eigen-hitter

## 1. INTRODUCTION

Graphs and networks are suitable descriptors of various real-life environments like social activity, professional collaboration, web activity, etc. [12, 16]. They reflect local and global relationships between the objects, which they model.

---

\*Corresponding author.

Studying how these objects interact with each other is useful from different perspectives. A graph is complete if all of its objects are connected to each other [5]. We are often interested to find out whether a graph is complete or not. Verifying the completeness of a graph consumes quadratic space and time with respect to its order. Considering the massive growth and dynamics of real-life networks, this becomes time/space inefficient. Therefore, designing sublinear algorithms is very important in massive data analytics [15].

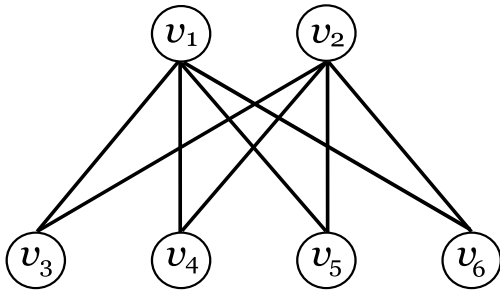
Due to the explosive growth of volume of the real-life datasets (the emergence of *big data*), many of the computational problems have been redefined to overcome the bottlenecks of time/space complexity. In this paper, we readdress the problem of verifying the completeness of a graph in a streaming model. In streaming models, the data are available as a sequence of items (stream) and the data cannot be stored entirely [20]. Therefore, we have to examine the data within a few passes (may be single) as the available memory is also limited. Again, the processing time per item has to be sublinear. This imposes a new kind of uncertainty in computing beyond approximation and randomization.

Here, we consider that the adjacency matrix of a graph is available as a stream. Adopting a turnstile model, we estimate completeness of the corresponding graph based on the  $\ell_1$  norm. Initially, we study the problem for a bipartite graph in the streaming model and generalize it further for any arbitrary simple graph. We employ some recent approximation results on  $\ell_1$  heavy eigen-hitters to find out top  $k$  eigen values, respectively [3]. The proposed algorithms run in linear time and consumes space proportional to  $k^2$  and the error parameters. We also demonstrate the effectiveness of the approaches in analyzing social and other real-life networks.

The current paper is organized as follows. Some background details and motivating applications are included in section 2 and section 3, respectively. Section 4 describes the state-of-the-art. Some theoretical results are provided in section 5 and based on this the proposed method is presented in section 6. Section 7 and section 8 cover some empirical results and discussions. Finally, section 9 concludes the paper.

## 2. PRELIMINARIES

Let us introduce some formal notations and standard definitions that will be used throughout the paper. We assume that  $|S|$  denotes the size (cardinality) of a set  $S$ . A graph is a doublet  $G = (V, E)$ , where  $V$  denotes the set of vertices and  $E \subseteq V \times V$  denotes the set of edges. The term *graph*



**Figure 1: A complete bipartite graph with the sets of disjoint vertices  $\{v_1, v_2\}$  and  $\{v_3, v_4, v_5, v_6\}$ .**

is used to refer to a simple graph (without self-loops or parallel edges [7]) that is undirected and labeled. Suppose the adjacency matrix of a graph  $G$  is denoted as  $A_G$ . A subgraph of a graph contains a subset of the vertices and edges. A subgraph is said to be induced by a vertex set if it has exactly the edges that appear in the original graph over the same vertex set. A graph is complete if all of its vertices are connected to each other, i.e.  $E = V \times V$ . A clique is a complete subgraph (often restricted to be maximal) of a graph [5]. A graph is said to be bipartite if its vertices can be segregated into two disjoint subsets, say  $V_1, V_2$ , such that  $V_1 \cap V_2 = \phi$ ,  $V_1 \cup V_2 = V$  and  $E \subseteq V_1 \times V_2$ . A complete bipartite graph has exactly  $|V_1| \times |V_2|$  edges (see Fig. 1). The other notations and graph-theoretic terminologies have their usual meaning, unless specified otherwise.

In this study, we assume that graphs are available under a streaming setting. In conventional data streaming models, the input stream  $\langle s_1, s_2, \dots \rangle$  arrives sequentially (item-wise) and describes an underlying signal [11]. The streaming models vary one from the other depending upon how the  $s_i$ 's represent the signal. Here, we consider a turnstile model where the underlying signal  $S$  is a one-dimensional function  $S : [1 \dots N] \rightarrow R$ ,  $R$  denoting the real space, where the  $s_i$ 's are updates to  $S[j]$ 's [20]. Note that in case of streaming graphs, represented as a real symmetric adjacency matrix, we obtain a strict turnstile model by default where  $S[j]$ 's are always non-negative. Inspired from the earlier formalizations [9], we define a streaming graph in a strict turnstile model as follows.

*Definition 1. A streaming graph in a strict turnstile model is a simple graph on  $n$  vertices  $V = \{v_1, v_2, \dots, v_n\}$  with edges  $E = \{(v_i, v_j) : s_k = (i, j) \text{ for some } k \in [m]\}$ , where the data items  $s_k \in [n] \times [n]$  are available as an input stream  $\mathcal{S} = \langle s_1, s_2, \dots, s_m \rangle$  pursuing a strict turnstile model.*

In this paper, we address a stronger version of the streaming problems involving linear time and sublinear space. To formalize, we would like the per-item processing time, storage and overall computing time to be simultaneously  $O(N, t)$ , preferably *polylog*( $N, t$ ), at any time instant  $t$  in the data stream. A sketch is often necessary to map the original space to a reduced space, retaining the necessary properties, to achieve this. We formally define a sketch as follows.

*Definition 2. A sketch  $\psi$  of a data set  $x$ , with respect to some function  $f$ , is a projection of  $x \rightarrow \psi$  from which one can compute  $f(x)$ .*

Our proposed algorithms and the related theoretical results are mainly based on approximating the heavy eigen-hitters in a streaming graph. We include the definition of heavy eigen-hitters below.

*Definition 3. The  $\phi$ -heavy eigen-hitters of a graph  $G$  are the eigen values that are at least  $\phi$ -fraction of the total mass of all the eigen values of the matrix  $A_G$ .*

In Definition 3, the total mass of eigen values represents, in a simpler understanding, the summation of all the eigen values. A norm is a function that assigns a strictly positive length (or size) to each vector in a vector space, other than the zero vectors (having a length zero). In general, we define the  $\ell_p$  norm as follows.

*Definition 4. For any non-zero vector  $x$ , the  $\ell_p$  norm is defined as*

$$\|x\|_p = \left( \sum_i^n |x^i|^p \right)^{1/p}, \quad (1)$$

where  $p \geq 1$  denotes a real constant.

*Definition 5. The  $\ell_1$  heavy eigen-hitters are the heavy eigen-hitter values based on the total mass in  $\ell_1$  norm.*

We discuss some real-life applications of estimating completeness in a streaming graph in the subsequent section.

### 3. MOTIVATING APPLICATIONS

Recent efforts in analyzing the available massive volume of data have assisted in both productivity growth and innovation in the industry and academia. It has immense potential in understanding the World Wide Web (WWW), financial sectors, medical analytics, public service domains, etc. Graphs can highlight large-scale global relations in effective ways for streaming data. Therefore, many futuristic applications are addressable with graph problems. Our target problem of estimating the completeness of a graph in a streaming model is also of high importance. We foresee a number of applications of this problem in various emerging areas of *big data*. Three of these are highlighted below.

- **Social network analysis:** Social communication at large-scale, rooted in WWW, has enabled the modeling of trillions of interactions between various social groups (e.g., researchers, students, actors, etc.). For the last decade or so, there is a massive growth of un-analyzed data in this area. Various other social communication methods like social networking websites (Facebook, Twitter, etc.), smart phones, multimedia applications, etc. are also contributing to these growing volumes of networked data. This increases the amount of dataflow per unit time and area. In accordance with this growth of data, analyses started with representing a network as a graph where the vertices are the elements and the edges denote their relations. Studying such large-scale graphs and their topologies might provide important features about the participating elements. Analyzing the dynamics of social networks is also interesting from different perspectives. Completeness can be verified for a portion of the streaming data so as to ensure whether the corresponding set of vertices (or a subgraph) arriving currently is

forming a clique or not. Again, the completeness of bipartite graphs may reveal interactions at the maximum scale between two different social groups. These are very important in a streaming setting. A recent attempt has shown that spreading rumors in real-life social networks is (surprisingly) faster than in complete graphs [8]. Therefore, studying graph completeness is also important for benchmarking analyses.

- **Analysis of biological networks:** Biological systems are often modeled as a network of biomolecules for understanding their cooperative activity. With the advancements of high-throughput technologies, enormous amount of experimental data is becoming accessible day by day. Biological networks may not be available as a stream but analyzing networks in linear/sublinear time/space is useful for dynamic sequencing of genes or for studying protein-protein interactions. Estimating completeness in such large-scale biological networks might help in prompt identification of strongly connected components. In a spreading disease network, biomolecules get rapidly affected by one another and such behaviors can be analyzed with completeness verification models. The broader goal is certainly to facilitate the system level understanding of cell-to-cellular components and its subprocesses.
- **Studying communication networks:** The number of communication service providers has rapidly increased over the last few decades around the world. Their growth has not only increased the volume of data but also its variability. This type of data can also be modeled as a network to understand various properties. Analyzing such networks might help the service providers to decide whether they should involve new services. Estimating completeness in such networks will be helpful in identifying the saturation of connectivity. This will invoke the demand of new communication providers. Again for the better understanding of a dynamic setting, we need to study the communication networks modeled on streaming graphs.

In the following section, we discuss the state-of-the-art of finding completeness of graphs, estimating cliques and the related progresses in streaming algorithms.

## 4. RELATED WORKS

Verifying the completeness of a graph is a special case of clique problems. A graph is complete if its clique number is of the order of the graph. Finding the maximum order clique in a graph is known to be an NP-hard problem [5]. An important study about a decade ago showed that the approximation of a maximal clique in polynomial time is hard within a factor of  $n^{1-\epsilon}$  (for any  $\epsilon > 0$ ), unless  $\text{NP} = \text{ZPP}$  (where  $n$  is the number of vertices in the graph) [14]. ZPP stands for zero-error probabilistic polynomial time. The problems in ZPP can be exactly solved in expected polynomial time by a probabilistic algorithm. It is strongly believed that  $\text{ZPP} \subset \text{NP}$  and the hypothesis  $\text{NP} \neq \text{ZPP}$  is almost as strong as  $\text{P} \neq \text{NP}$  [14], where P denotes the decision problems solvable in polynomial time using a deterministic Turing machine. For this reason, many of the recent algorithms to solve the maximum clique problem (MCP) are based on metaheuristic approaches [4]. To approximate cliques, spectral approaches

showed promise in earlier studies. Spectral graph theory is also important in analyzing the bounds of completeness. A few attempts were made earlier to estimate the chromatic number of a graph using eigen values [22], which can be further related with the clique number of a graph. Based on eigen value computations, several upper bounds on the clique number were derived previously [2]. Recently, these bounds (and also lower bounds) were tightened further [6]. Current studies indicate that a relation with the spectral radius with the clique might help us to estimate the upper bound of the clique in a streaming model [17].

Streaming algorithms have been in focus for more than a decade. But this domain is still in a nascent stage. The limited earlier contributions before 2005 have been well reviewed in [20]. While presenting this survey, Muthukrishnan also addressed some real life problems based on streaming models. Following this, diverse efforts were made to revisit and solve a number of problems in a streaming setting. There were studies on matrix approximation, matrix decomposition, low rank approximation,  $\ell_p$  regression, etc. [13, 17, 18]. There has been an influential line of work on computing a low-rank approximation of a given matrix, starting with the works of [10, 21]. A lot of works were done on linear algebra in a streaming model. Also low rank approximation made the analysis of massive data less complicated. Very recently, the  $\ell_1$  and  $\ell_2$  heavy eigen-hitter problems have been estimated in the streaming model in a lower dimension [3]. Notably, the heavy eigen-hitters problem was first proposed in [20]. Andoni and Huy achieved a success probability of  $\frac{5}{9}$  [3]. They also estimated the residual error with the same probabilistic accuracy. Sampling and sketching methods for producing low-complexity approximations of large matrices is in focus for the last few decades. We estimated the completeness of a graph in the streaming model based on the computations of  $\ell_1$  heavy eigen-hitters.

## 5. THEORETICAL RESULTS

In this section, we present some useful theoretical outcomes and derive some new results that will be helpful in devising the proposed algorithms. Let  $A_G$  be a real symmetric  $n \times n$  ( $n \geq 1$ ) matrix denoting the adjacency relations in a graph  $G$ . Further assume  $\lambda_i(A_G)$  be the  $i^{\text{th}}$  largest eigen value of  $A_G$  in absolute value. Now, if  $\psi$  represents a sketch of the matrix  $A_G$  where  $\psi = PA_G P^T$ . Then, we have the following important result from a recent study [3].

**THEOREM 1.** *There is a linear sketch of the real symmetric matrix  $A_G$ , of dimension  $n \times n$ , using space  $O(k^2 \epsilon^{-4})$  ( $\epsilon > 0$ ,  $k \in \{1, 2, \dots, n\}$ ), from which one can produce values  $\tilde{\lambda}_i$ , for  $i \in [k]$ , satisfying the following with at least  $\frac{5}{9}$  success probability*

$$|\lambda_i(A_G) - \tilde{\lambda}_i| \leq \epsilon |\lambda_i(A_G)| + \frac{1}{k} S_1^{k+1},$$

where  $S_1^{k+1} = \sum_{i>k} |\lambda_i(A_G)|$  denotes the residual “ $\ell_1$  error”.

Now, we derive the following result on bipartite graphs using the previous claim in Theorem 1.

**THEOREM 2.** *On fixing a value of  $\epsilon > 0$ , one can ensure whether  $G$  is a complete bipartite graph by deriving a linear sketch  $\psi$  from  $A_G$  whose top two heavy eigen-hitters in absolute value should be the same satisfying*

$$\lambda_1(\psi) = (1 \pm \epsilon) \lambda_1(A_G) \pm S_1^2,$$

and

$$\lambda_2(\psi) = (1 \pm \epsilon)\lambda_2(A_G) \pm 0.5S_1^3,$$

and the third largest eigen value satisfies

$$\lambda_3(\psi) = \pm 0.3S_1^4.$$

PROOF. The eigen values of a complete bipartite graph  $G$  can be ordered as  $\{\lambda_1(A_G), 0, \dots, 0, \lambda_n(A_G)\}$ , where  $\lambda_1(A_G) = -\lambda_n(A_G) = \lambda$  (say) [2]. Therefore, if we obtain a decreasing order of the eigen values of  $A_G$  in absolute value, we would get  $\{\lambda, \lambda, 0, \dots, 0\}$ . Since  $G$  does not contain any self-loops, the trace of  $A_G$  should be zero. Then, we can write

$$\sum_{i=1}^n \lambda_i(A_G) = 0.$$

It is understandable that if the eigen values are decreasingly ordered by absolute value, say  $\{\lambda'_1(A_G), \lambda'_2(A_G), \dots, \lambda'_n(A_G)\}$ , and if  $\lambda'_1(A_G) = \lambda'_2(A_G)$  and  $\lambda'_3(A_G) = 0$ , then rest of the eigen values of  $A_G$  will be certainly zero. This is because the rest of the eigen values cannot be negative (being in absolute value) and no more than zero (being in decreasing order). So, it is sufficient for  $A_G$ , to have the first two largest eigen values same in absolute value and the third one zero, for claiming that the corresponding graph  $G$  is complete bipartite. Now, from Theorem 1, we can derive that the first  $k$  eigen values, for a particular  $\epsilon > 0$ , will satisfy the following for a linear sketch  $\psi$

$$\lambda_i(\psi) = (1 \pm \epsilon)\lambda_i(A_G) \pm \frac{1}{k}S_1^{k+1}, \quad (2)$$

Using Eqn. (2), one can verify whether the first two largest eigen values are same and estimate their values from the sketch  $\psi$  satisfying

$$\begin{aligned} \lambda_1(\psi) &= (1 \pm \epsilon)\lambda_1(A_G) \pm \frac{1}{1}S_1^{1+1} \\ \implies \lambda_1(\psi) &= (1 \pm \epsilon)\lambda_1(A_G) \pm S_1^2. \end{aligned}$$

and similarly

$$\begin{aligned} \lambda_2(\psi) &= (1 \pm \epsilon)\lambda_2(A_G) \pm \frac{1}{2}S_1^{2+1} \\ \implies \lambda_2(\psi) &= (1 \pm \epsilon)\lambda_2(A_G) \pm 0.5S_1^3. \end{aligned}$$

Again, one can verify whether the third largest eigen value is zero and estimate its value from the sketch  $\psi$  satisfying

$$\begin{aligned} \lambda_3(\psi) &= (1 \pm \epsilon)\lambda_3(A_G) \pm \frac{1}{3}S_1^{3+1} \\ \implies \lambda_3(\psi) &= (1 \pm \epsilon).0 \pm 0.3S_1^4 \\ \implies \lambda_3(\psi) &= \pm 0.3S_1^4. \end{aligned}$$

This altogether completes the required proof.  $\square$

**THEOREM 3.** *On fixing a value of  $\epsilon > 0$ , one can ensure whether  $G$  is a complete graph by deriving a linear sketch  $\psi$  from  $A_G$  whose top two heavy eigen-hitters in absolute value satisfy the following*

$$\lambda_1(\psi) = (1 \pm \epsilon)(n-1) \pm S_1^2.$$

and

$$\lambda_2(\psi) = (\epsilon \pm 1) \pm 0.5S_1^3.$$

PROOF. The eigen values of a complete graph  $G$  can be ordered as  $\{n-1, -1, \dots, -1\}$  [2]. Therefore, if we obtain a decreasing order of the eigen values of  $A_G$  in absolute value, we would get  $\{n-1, 1, \dots, 1\}$ . Since  $G$  does not contain any self-loops, the trace of  $A_G$  should be zero. Then, we can write

$$\sum_{i=1}^n \lambda_i(A_G) = 0.$$

It is understandable that if the eigen values are decreasingly ordered by absolute value, say  $\{\lambda'_1(A_G), \lambda'_2(A_G), \dots, \lambda'_n(A_G)\}$ , and if  $\lambda'_1(A_G) = n-1$  and  $\lambda'_2(A_G) = 1$ , then certainly the rest of the eigen values of  $A_G$  should also be one. This is because the rest of the eigen values cannot be negative (being in absolute value) and no more than one (being in decreasing order). So, it is sufficient for  $A_G$ , to have the first two largest eigen values as  $n-1$  and one, respectively, for claiming that the corresponding graph  $G$  is complete. We have already derived that the first  $k$  eigen values, for a particular  $\epsilon > 0$ , will satisfy Eqn. (2) for a linear sketch  $\psi$ . Then using this, the first largest eigen value can be estimated as

$$\begin{aligned} \lambda_1(\psi) &= (1 \pm \epsilon).\lambda_1(A_G) \pm \frac{1}{1}S_1^{1+1} \\ \implies \lambda_1(\psi) &= (1 \pm \epsilon).(n-1) \pm S_1^2. \end{aligned}$$

and the second largest eigen value can be estimated as

$$\begin{aligned} \lambda_2(\psi) &= (1 \pm \epsilon).\lambda_2(A_G) \pm \frac{1}{2}S_1^{2+1} \\ \implies \lambda_2(\psi) &= (1 \pm \epsilon).(-1) \pm 0.5S_1^3 \\ \implies \lambda_2(\psi) &= (\epsilon \pm 1) \pm 0.5S_1^3. \end{aligned}$$

This altogether completes the required proof.  $\square$

In the next section, we present our approaches for completeness verification of bipartite graphs and any arbitrary graph in a streaming setting.

## 6. PROPOSED METHOD

Our algorithms are principally based on the concept of generating a projection  $P$ , of size  $O(k/\epsilon^2)$  by  $n$ , and computing the sketch  $\psi = PA_G P^T$  for  $\ell_1$  to retain (and thus estimate) the properties of the heavy eigen-hitters as close as possible. This saves the space requirements and computational time together. The method of verifying completeness of bipartite graphs is formally presented as Algorithm 1. Theorem 2 serves as the base of this approach. Note that, for estimating the completeness of bipartite graphs, we require the top two eigen values in absolute value to be same in original matrix  $A_G$ , i.e.  $\lambda_1(A_G) = \lambda_2(A_G) = \lambda$  (say). Therefore, the eigen value differences in the sketch matrix  $\psi$  result into the following relation (using Theorem 2).

$$\lambda_1(\psi) - \lambda_2(\psi) = \pm 2\epsilon\lambda \pm S_1^2 \pm 0.5S_1^3.$$

This provides an error estimation of the relation derived in Theorem 2.

In Algorithm 2, we present the formal approach of verifying completeness of any arbitrary graph. This is a more generalized approach with a fewer number of eigen value estimations. We mainly use the results from Theorem 3 to devise this algorithm. In both the algorithms described, for



---

**Algorithm 1** An algorithm for estimating completeness of bipartite graphs

---

**Input:** The adjacency matrix  $A_G$  of the bipartite graph  $G$ .

**Output:** The decision about the completeness of  $G$ .

**Algorithmic Steps:**

- 1: Obtain a sketch  $\psi = PA_G P^T$ , where  $P$  is a  $t \times n$  matrix with  $\Theta(\frac{\log^2 n}{\epsilon^2})$ -wise independent entries identically distributed as  $N(0, \frac{1}{t})$ .
  - 2: Compute the top three largest eigen values of  $\psi$  in the decreasing order denoted as  $\lambda_1(\psi)$ ,  $\lambda_2(\psi)$  and  $\lambda_3(\psi)$ , respectively.
  - 3: **if**  $\lambda_1(\psi) = \lambda_2(\psi)$  and  $\lambda_3(\psi) = \pm 0.3S_1^4$  **then**
  - 4:  $G$  is a complete bipartite graph.
  - 5: **end if**
- 

bipartite and general graphs, the  $\Theta(\frac{\log^2 n}{\epsilon^2})$ -wise independent entries for the random matrix  $P$  are generated following an earlier approach [1].

---

**Algorithm 2** An algorithm for estimating completeness of any arbitrary graph

---

**Input:** The adjacency matrix  $A_G$  of the graph  $G$ .

**Output:** The decision about the completeness of  $G$ .

**Algorithmic Steps:**

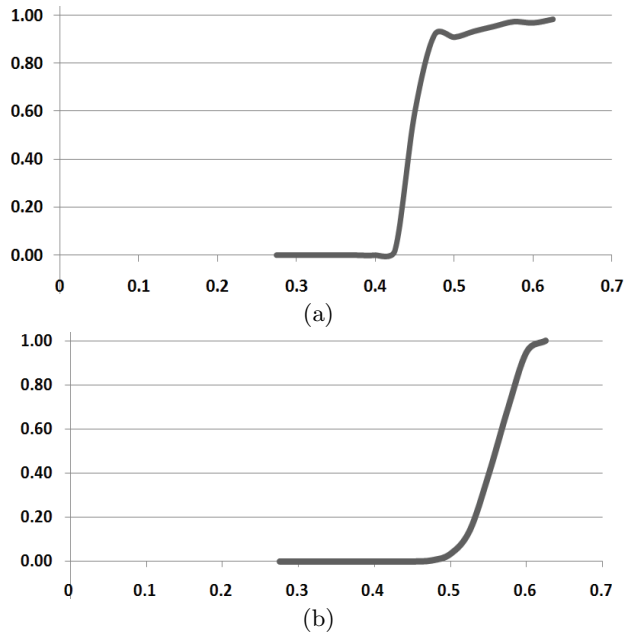
- 1: Obtain a sketch  $\psi = PA_G P^T$ , where  $P$  is a  $t \times n$  matrix with  $\Theta(\frac{\log^2 n}{\epsilon^2})$ -wise independent entries identically distributed as  $N(0, \frac{1}{t})$ .
  - 2: Compute the top two largest eigen values of  $\psi$  in the decreasing order denoted as  $\lambda_1(\psi)$  and  $\lambda_2(\psi)$ , respectively.
  - 3: **if**  $\lambda_1(\psi) = (1 \pm \epsilon)(n-1) \pm S_1^2$  and  $\lambda_2(\psi) = (\epsilon \pm 1) \pm 0.5S_1^3$  **then**
  - 4:  $G$  is a complete graph.
  - 5: **end if**
- 

## 7. EMPIRICAL STUDY

We considered two real-life networks for testing the outcome of the proposed algorithms. The algorithms were implemented in MATLAB and the simulations were performed on an HP Laptop with Intel(R) Core(TM) i5-2410M processor running at 2.30 GHz speed and having 4 GB primary memory. one of these networks is a complete bipartite graph and the other one is sparse. The experimental procedures are briefly discussed below.

### 7.1 Study on Synthetic Networks

We have constructed two synthetic networks, one complete bipartite and another complete network, both having orders 40 for performance analysis of the proposed approaches. The complete bipartite network has equal number of partitions. In both these cases, dimension of the sketch matrix becomes  $t \times 40$ . We have varied  $t$  from 10 to 25 and several arbitrary matrices are generated by employing random selection method on a normal distribution (identically) with parameters  $(0, 0.01)$ . Finally, the eigen values are estimated (using Algorithm 1 and Algorithm 2) and compared with the original values. The obtained eigen values indicate their completeness. The Figs. 2(a-b) show the accuracies of the eigen values against the difference of dimensions between the sketch and the original matrix. It becomes clear



**Figure 2:** The average accuracy obtained against the sketch difference of the synthetic (a) complete bipartite network and (b) complete network.

that the performance rapidly improves after certain threshold. So, proper selection of the dimension of the sketch vector is very much important.

### 7.2 Study on Social Networks

We have used a large-scale social interaction data of Facebook, consisting of ‘circles’ (denoting ‘friends lists’), from a recent study [19]. This interaction data is used to construct a large undirected unweighted social network having 4039 vertices and 88234 edges. The average clustering coefficient of the network is found to be 0.61, establishing that it is not complete. We have analyzed this and computed a sketch of dimension  $100 \times 4039$  with elements identically distributed in  $N(0, 0.01)$ . Finally, Algorithm 2 is applied on this. The obtained eigen values are found to be quite far from the values supporting its completeness (as per Theorem 3).

## 8. DISCUSSION

The approaches to completeness verification presented in the current paper is important from two different perspectives. First, the theoretical results provided might be useful in estimating the clique number of a graph that depends on the number of eigen values no greater than ‘-1’ [2]. Secondly, the implementation details might be useful in developing many other algorithms that work in a streaming setting. Our assumption of a strict turnstile model does not weaken the results because the problem demands so. The real symmetric form of adjacency matrices of streaming graphs can be well captured using a turnstile model. Our attempts of utilizing heavy eigen-hitters are also very promising. The approaches to standard vector heavy hitters return the elements that are most frequent (heavy coordinates) [20]. On the contrary, our methods do not find the elements that are heavy hitters. This saves an additional factor of  $O(\log n)$  to the space requirements (ignoring the random seed size).

Another significant advantage of the proposed algorithms is that their performances are independent of the seed selection for generating random matrices. It might appear that the algorithms work only on static graphs (i.e. on a fixed adjacency matrix) to compute the sketches. But they also work in a streaming setting because the algorithms, being linear in computational time, are also capable of supporting arbitrary updates to the matrix. The major limitation of our approaches is that the success probabilities are still low since they rely on multiple applications of Theorem 1. Again, it might be criticized that large real-life networks, like social entities and their interactions, are rarely complete. But the approaches of estimating completeness are still applicable where time is a major constraint. Therefore, the proposed methods are very much generalized.

## 9. CONCLUSION

In this paper, we have provided the first approximation algorithms for estimating the completeness of bipartite graphs and, in general, any arbitrary graph. Our results are promising and useful for diverse applications. We have also implemented the proposed algorithms and verified results on two test cases. The approaches are promising for many further directions of research on big data at a network level. However, the success probabilities of our algorithms are still poor that we would like to improve in near future. Again, we wish to extend the current analysis using property testing.

## 10. ACKNOWLEDGMENTS

The authors are thankful to Huy L. Nguyen in the department of Computer Science of Princeton University for his important feedback over an initial draft of the paper.

## 11. REFERENCES

- [1] N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple constructions of almost  $k$ -wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.
- [2] A. T. Amin and S. L. Hakimi. Upper bounds on the order of a clique of a graph. *SIAM Journal on Applied Mathematics*, 22(4):569–573, 1972.
- [3] A. Andoni and H. L. Nguyen. Eigenvalues of a matrix in a streaming model. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1729–1737, New Orleans, USA, 2013.
- [4] S. Bandyopadhyay and M. Bhattacharyya. Mining the largest dense vertexlet in a weighted scale-free graph. *Fundamenta Informaticae*, 96:1–25, 2009.
- [5] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In D. Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization: Supplementary Volume A*, pages 1–74. Kluwer Academic, Dordrecht, 1999.
- [6] M. Budinich. Exact bounds on the order of the maximum clique of a graph. *Discrete Applied Mathematics*, 127:535–543, 2003.
- [7] R. Diestel. *Graph Theory*. Springer-Verlag Heidelberg, New York, 2005.
- [8] B. Doerr, M. Fouz, and T. Friedrich. Why rumors spread fast in social networks. *Communications of the ACM*, 55(6):70–75, 2012.
- [9] J. Feigenbaum, S. Kannan, A. McGregor, and S. Suri. Graph distances in the data stream model. *SIAM Journal of Computing*, 38(5):1709–1727, 2008.
- [10] A. Frieze, R. Kannan, and S. Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004.
- [11] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One pass summaries for approximate aggregate queries. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 79–88, San Francisco, CA, USA, 2001.
- [12] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences USA*, 99:7821–7826, 2002.
- [13] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.
- [14] J. Håstad. Clique is hard to approximate within  $n^{1-\epsilon}$ . *Acta Mathematica*, 182(1):105–142, 1999.
- [15] P. Indyk, R. Levi, and R. Rubinfeld. Approximating and testing  $k$ -histogram distributions in sub-linear time. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 15–22, Scottsdale, Arizona, 2012.
- [16] H. Ino, M. Kudo, and A. Nakamura. Partitioning of web graphs by community topology. In *Proceedings of the 14th International World Wide Web Conference*, pages 661–669, Chiba, Japan, 2005.
- [17] R. Kannan and S. Vempala. Spectral algorithms. *Foundations and Trends in Theoretical Computer Science*, 4(3-4):157–288, 2009.
- [18] M. W. Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends in Machine Learning*, 3(2):123–224, 2011.
- [19] J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Proceedings of the Neural Information Processing Systems*, pages 548–556, 2012.
- [20] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.
- [21] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: a probabilistic analysis. *Journal of Computer and System Sciences*, 61(2):217–235, 2000.
- [22] H. S. Wilf. The eigenvalues of a graph and its chromatic number. *Journal of the London Mathematical Society*, 42(1):330–332, 1967.

# Mining Urban Data (MUD) Gennady Andrienko (Fraunhofer

Institute IAIS, Germany)

Dimitrios Gunopulos (University of Athens, Greece)

Vana Kalogeraki (Athens University of Economics and Business, Greece)

Ioannis Katakis (National and Kapodistrian University of Athens, Greece)

Pedro José Marrón (University Duisburg-Essen, Germany)

Katharina Morik (Technischen Universität Dortmund, Germany)

Olivier Verscheure (IBM Research, Ireland)

# Mining Trajectory Data for Discovering Communities of Moving Objects

Corrado Loglisci  
Department of Computer Science  
University of Bari "Aldo Moro"  
Bari, Italy  
corrado.loglisci@uniba.it

Donato Malerba  
Department of Computer Science  
University of Bari "Aldo Moro"  
Bari, Italy  
donato.malerba@uniba.it

Apostolos N. Papadopoulos  
Department of Informatics  
Aristotle University  
Thessaloniki, Greece  
papadopo@csd.auth.gr

## ABSTRACT

Recent advances on tracking technologies enable the collection of spatio-temporal data in the form of trajectories. The analysis of such data can convey knowledge in prominent applications, and mining groups of moving objects turns out to be a valuable mean to model their movement. Existing approaches pay particular attention in groups where objects are close and move together or follow similar trajectories by assuming that movement cannot change over time. Instead, we observe that groups can be of interest also when objects are spatially distant and have different but inter-related movements: objects can start from different places and join together to move towards a common location. To take into account inter-related movements, we have to analyze the objects jointly, follow their respective movements and consider changes of movements over time. Motivated by this, we introduce the notion of *communities* and propose a computational solution to discover them. The method is structured in three steps. The first step performs a feature extraction technique to elicit the inter-related movements between the objects. The second one leverages a tree-structure in order to group objects with similar inter-related movements. In the third step, these groupings are used to mine communities as groups of objects which exhibit inter-related movements over time. We evaluate our approach on real data-sets and compare it with existing algorithms.

## Categories and Subject Descriptors

H.2.8 [Database Applications]: Data mining

## Keywords

Trajectories, Mining, Groups of Moving Objects.

## 1. INTRODUCTION

The tremendous advances in positioning technologies, such as telemetry, GPS equipment and smart mobile phones, have enabled tracking of any type of moving objects and collecting spatio-temporal data into growing repositories. Some

example applications follow: *i*) In location-based social networks, people travel in the real world and leave their location history in the form of a *trajectory*. These trajectories do not only connect locations in the physical world but also bridge the gap between people and locations [12]. *ii*) GPS devices allow the recording of vehicle locations [11]. Such information often includes data for human mobility. *iii*) Zoologists are investigating the impact of the levels of urbanization on the migration, distribution and habitat use of animals [9].

In the aforementioned applications, one can be interested in the discovery of groups of objects which move together or in a similar manner. For instance, in car pooling it could be useful to determine people with the same route to share the car. Such problems are not novel in the literature [5] and most of the efforts result in mining groups of moving objects, such as *flocks* [1], *convoys* [4] and *swarms* [8].

The spatio-temporal properties of these groups is the main distinguishing aspect. In particular, a flock contains at least  $m$  objects moving in the same direction within a circular region with a user-defined radius. Variants of the flock include also a notion of time-interval (with minimum duration defined by the user) according to which in each time-stamp of the interval a disc containing  $m$  objects can be identified. The rigid characteristic of fixed circular shape could miss some groups of arbitrary form.

The introduction of the notion of *density* avoids this drawback and allows to discover groups, named as convoys, which have no limitations on the shape and size. A convoy is defined as a cluster of objects and it is identified by means of a density-based clustering technique which checks for the condition of density-connectedness on the objects and for all the time-stamps of a time-interval [4].

A more general group type is represented by the swarm concept, which, in contrast to flocks and convoys, it is not required to hold for all time-stamps of a time-interval, but it can occur more sporadically. In the classical notion of swarm this temporal constraint corresponds to a minimum number of time-stamps which are not necessarily consecutive.

**Motivation.** The algorithms to detect flocks, convoys and swarms are designed to capture similarities among (sub)trajectories but leave unexplored two interrelated aspects which instead appear to be new sources of information to exploit: *i*) movements may depend on each other and may hide interactions among the objects, *ii*) movements can reflect changes of the motion of the objects and implicitly denote their dynamic behaviour. Interactions reflect the possible relationships in which the objects can be involved in space and time, they can provide a more complete description of the groups by explaining even the cause of their formation. Interactions can evolve because the objects can move near each

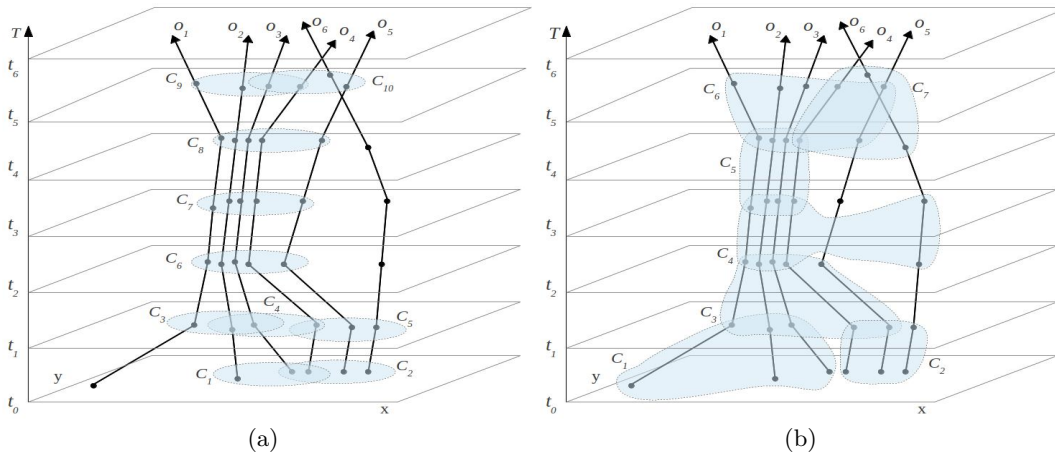


Figure 1: Moving objects grouped using: (a) flocks, convoys and swarms and (b) communities.

other and then move away. Indeed, moving objects intrinsically are dynamic, their motion is not necessarily linear and it can be influenced by the properties or needs of each object and by the interactions with other objects as well. For instance, in social studies, we can observe individuals which begin to move from different locations, they could come near until to join together in proximity of a point of interest, they could remain there for a time and then go away from each other. So, those individuals can be members of a group even without having followed similar trajectories.

In that kind of problems, a group can turn out to be interesting not only when its members are spatially close and move similarly, but also when they are far apart and have different but inter-related movements, or also when they have different movements but are involved in the same type of interactions. Existing approaches are not prepared to handle this concept, mainly due to the following reasons:

- Most of the existing techniques rely on a static group concept where objects have to always meet the same spatio-temporal properties: for instance, the members of a group are required to be close each other in each time-stamp.
- The trajectory corresponds to a geometric abstraction of the movement and is defined as a series of punctual time-stamped observations that cannot indicate neither how the object moves over time nor whether there exists any form of relationship with other trajectories.

**Related Work and Contributions.** In this paper, we introduce the concept of *community* based on the concepts of interaction among the objects and change in the movements of the objects. The interaction between two objects  $o_i$  and  $o_j$  is defined on the basis of the movement that an object  $o_i$  performs with respect to another object  $o_j$ , while the change concerns the variations of spatio-temporal characteristics that can be observed in the movement of each object. Therefore, changes of an object's motion may influence or determine its interaction with other objects.

A community consists of a set of objects in common to a set of groups arranged in sequence. In its turn, a group contains  $n - 1$  pairs formed with objects taken from a set of  $n$  elements: a pair is formed with one object in common to all pairs (*reference object*) and the other object taken from

the remaining  $n - 1$  (*participants*). The pairs of a group exhibit very similar spatio-temporal features. Differently from flock, convoy and swarm, the timing of a community is based on time-intervals created from the time-stamps of the positions. We clarify the difference between a community and other group types in the following example.

Another notion of *community*, proposed in [10], models the similarities of moving objects in four information sources, namely semantic properties of the locations, temporal duration of the trajectory, spatial proximity and movement velocity. This notion anyway requires that the objects move similarly in all time-stamps whereas the result cannot include communities with discontinuities over time.

EXAMPLE 1. In Figure 1(a), six objects are tracked and have the positions in six time-stamps included in the time-intervals  $[t_0, t_1]$ ,  $[t_1, t_2]$ ,  $[t_2, t_3]$ ,  $[t_3, t_4]$ ,  $[t_4, t_5]$ ,  $[t_5, t_6]$ . Let  $k=3$  the minimum number of objects required for the final groups. Clusters  $\{C_3, C_6, C_7, C_8, C_9\}$  share the objects  $\{o_1, o_2, o_3\}$  which form a flock in  $[t_1, t_2]$ ,  $[t_2, t_3]$ ,  $[t_3, t_4]$ ,  $[t_4, t_5]$ ,  $[t_5, t_6]$ . The group  $\{o_1, o_2, o_3, o_4\}$  corresponds to a convoy if we consider the notion of density-connectedness on the clusters  $\{C_4, C_6, C_7, C_8, C_9\}$ . Finally, with the objects in common to the clusters  $\{C_2, C_5, C_{10}\}$  we have the swarm  $\{o_4, o_5, o_6\}$ . In Figure 1(b), we have two communities, namely  $\{o_1, o_2, o_3\}$  and  $\{o_4, o_5, o_6\}$  respectively. The first one is composed of the objects in common to the sequence of the groups  $\{C_1, C_3, C_4, C_5, C_6\}$ , where the group  $C_1$  is collocated into the time-interval  $[t_0, t_2]$ ,  $C_3$  is collocated into the time-interval  $[t_1, t_3]$ ,  $C_4$  is collocated into  $[t_2, t_4]$ ,  $C_5$  is associated with  $[t_4, t_6]$ . The group  $C_1$  is composed of the pairs  $(o_2, o_1)$  (where  $o_2$  is the reference,  $o_1$  is the participant) and  $(o_2, o_3)$  (where  $o_2$  is the reference,  $o_3$  is the participant). The other groups can be interpreted in the same manner. The motions of the pairs  $(o_2, o_1)$  and  $(o_2, o_3)$  tells us that they start far apart and tend to move near while observing a variation of the mutual distance (in  $[t_0, t_3]$ ), then, they move together without any variation of the distance (in  $[t_2, t_5]$ ), finally they move apart (in  $[t_4, t_6]$ ). The community  $\{o_4, o_5, o_6\}$  is obtained from the non-consecutive groups  $\{C_2, C_7\}$ : the first group is collocated into  $[t_0, t_2]$ , the second group in  $[t_4, t_6]$ . In this community, the pairs  $(o_5, o_4)$  and  $(o_5, o_6)$  proceed by keeping the same distance in  $[t_0, t_2]$  while they exhibit a reduction of the mutual distance in  $[t_4, t_6]$ . ■

The previous example shows the difficulty of existing al-

gorithms to discover communities. Indeed, the algorithm for finding flocks is inadequate since it works with clusters in the strict form of a fixed disc. The method for detecting convoys cannot be used since it operates on the density-connectedness corresponding to the simultaneous application of conditions on the size and closeness for each cluster, which are criteria hard to be satisfied when considering distant objects. The difficulty of the algorithm for the discovery of swarms [8] lies in the accommodation of the temporal component and, specifically, in the fact that the members of the swarms are required to stick together for a number of possibly non-consecutive time-stamps. But this could mean having insignificant swarms characterized by completely disjointed time-stamps and fragmented movements. In summary, the contributions of this paper include:

- A new definition of group of moving objects which extends the classical notion of cluster based on the spatial closeness and density-connectedness.
- The exploitation of two new sources of information corresponding to the interactions among the objects and changes of their motions.
- The definition of spatio-temporal features able to model the interactions and changes of the movements of pairs of moving objects.
- The synthesis of a grouping technique which does not rely on a distance/dissimilarity measure.
- A performance evaluation and experimental comparison with existing techniques.

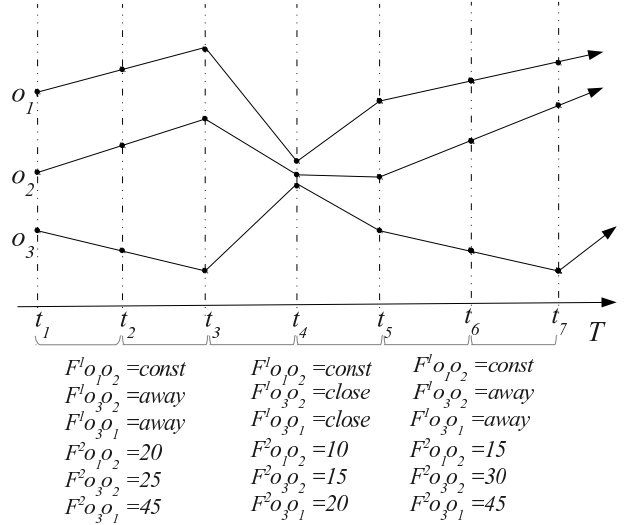
**Roadmap.** The remainder of this work is organized as follows. The next section presents some fundamental concepts related to our approach. Section 3 studies our proposal in detail. Performance evaluation results are offered in Section 4 whereas Section 5 concludes our work and discuss briefly future research directions.

## 2. FUNDAMENTAL CONCEPTS

In this section we present some fundamental concepts related to our proposal. Some frequently used symbols are given in Table 1. Let  $\mathcal{O}=\{o_1, o_2, \dots, o_n\}$  be the set of all moving objects and  $\mathcal{T} = \{\tau_1, \tau_2, \dots, \tau_m\}$  be the set of all time-stamps. The trajectory of an object  $o$  is a finite sequence of time-stamped locations denoted as  $t(o) : \langle (p_1, \tau_1), (p_2, \tau_2), \dots, (p_m, \tau_m) \rangle$  during the time-interval  $[\tau_1, \tau_m]$ , where

**Table 1: List of symbols.**

Symbol	Explanation
$\mathcal{O}$	all moving objects
$\mathcal{T}$	all time-stamps
$t(o)$ ( $t(o_u)$ )	trajectory of the object $o$ ( $o_u$ )
$\mathcal{F}$	set of descriptive features
$F_l$	$l$ -th features describing a pair of trajectories
$F_{o_u, o_v}^l$	value of the $l$ -th features for trajectories of $o_u, o_v$
$[\tau_1, \tau_m]$	time-interval containing time-stamps of $\mathcal{T}$
$\mathcal{G}$	pair group
$o_r$	reference object of a pair group
$o_s$	participant object of a pair group
$\mathcal{G}_f$	feature group
$\epsilon_{min_l}(\epsilon_{max_l})$	min (max) value of feature $F_l$
$e_l$	fixed value of the categoric feature $F_l$
$\mathcal{C}$	a community



**Figure 2: Feature generation from trajectories.**

$p_i \in \mathbb{R}^2$  is the geo-spatial position sampled at  $\tau_i \in \mathcal{T}$ . A trajectory may have time-stamps not necessarily equally distanced, they can be different from those of another trajectory as well as different trajectories may have different lengths (number of geo-spatial positions).

In this work, we do not analyze the original trajectories but we adopt a transformation technique which projects the trajectory data into a descriptive space which includes a finite set of features  $\mathcal{F}=\{F_1, \dots, F_l, \dots, F_f\}$  which are the real subject of our analysis. The features can take value in categoric or numeric domains. In particular, for each pair  $(o_u, o_v)$ , the transformation technique returns a set of valued features for the (sub)trajectories observed in two consecutive time-intervals, which we denote as  $[\tau_i, \tau_j] \cup [\tau_{j+1}, \tau_k]$  and name as *feature time-intervals*.

A simple illustration is reported in Figure 2. Consider the trajectories of three objects  $o_1, o_2, o_3$ . Let  $F_{o_u, o_v}^1, F_{o_u, o_v}^2$  be two features which describe the reciprocal movement between the objects  $o_u, o_v$  and their average mutual distance respectively. The domain of the feature  $F_{o_u, o_v}^1$  has categoric values {“const”, “far”, “close”} where “const” corresponds to two objects that travel together by keeping constant their distance, “away” corresponds to two objects that are moving away, and “close” corresponds to two objects that are moving closer. The domain of the feature  $F_{o_u, o_v}^2$  has numeric values in the set of natural numbers  $\mathbb{N}$ . The values of  $F^1$  and  $F^2$  are computed on the feature time-intervals  $[\tau_1, \tau_2] \cup [\tau_2, \tau_3]$ ,  $[\tau_3, \tau_4]$ ,  $[\tau_4, \tau_5]$ ,  $[\tau_5, \tau_6]$ ,  $[\tau_6, \tau_7]$ . So, for instance, the value of the feature  $F_{o_1, o_2}^1$  in the feature time-intervals  $[\tau_1, \tau_2] \cup [\tau_2, \tau_3]$  is “const”, while the value the feature  $F_{o_1, o_2}^2$  is 20. Figure 2 reports the remaining values of the features.

**DEFINITION 1 (PAIR GROUP).** *Given a subset of  $\mathcal{O}$  with  $m$  objects, a pair group  $\mathcal{G}$  consists of the  $(m-1)$  pairs of objects  $(o_r, o_s)$ , where  $r \in \{1, \dots, m\}, s = 1, \dots, m, r \neq s$ . The object  $o_r$  appears in all pairs and is named as reference object, while the objects  $o_s$  are named participant objects.*

**DEFINITION 2 (FEATURE GROUP).** *Given a pair group  $\mathcal{G}$ , the set of features  $\mathcal{F}=\{F_1, \dots, F_l, \dots, F_f\}$ , a feature group  $\mathcal{G}_f$  consists of the pairs of  $\mathcal{G}$  which, in the feature time-intervals  $[\tau_i, \tau_{i+k}] \subseteq \mathcal{T}$ ,  $[\tau_{i+k+1}, \tau_{i+2k}] \subseteq \mathcal{T}$ ,  $\dots$ ,  $[\tau_p, \tau_{p+k}] \subseteq \mathcal{T}$ ,  $[\tau_{p+k+1}, \tau_{p+2k}] \subseteq \mathcal{T}$ , satisfy the following conditions*

- $\forall (o_r, o_s) \in \mathcal{G}: \epsilon_{\min_i} \leq F_{o_r, o_s}^l < \epsilon_{\max_i}$ , iff  $F_i$  has numeric values,  
where  $\epsilon_{\min_i} \in \mathbb{R}$ ,  $\epsilon_{\max_i} \in \mathbb{R}$  are minimum and maximum values respectively for the feature  $F_i$ .
- $\forall (o_r, o_s) \in \mathcal{G}: F_{o_r, o_s}^l = \epsilon_i$ , iff  $F_i$  has categoric values,  
where  $\epsilon_i$  is a fixed value in the domain of  $F_i$ .

The values of  $\epsilon_{\min_i}$ ,  $\epsilon_{\max_i}$ ,  $\epsilon_i$  are specific for each feature group. The feature time-intervals have identical width and are arranged in chronological order.

Intuitively, a feature group is characterized by two components, one of nature geo-spatial, the other one of nature temporal. Definition 2 says that, in the time-intervals  $[\tau_i, \tau_{i+k}] \cup [\tau_{i+k+1}, \tau_{i+2k}], \dots, [\tau_p, \tau_{p+k}] \cup [\tau_{p+k+1}, \tau_{p+2k}]$ , the pairs of objects of  $\mathcal{G}$  have the same value for each categorical feature and the same range for each numeric feature. For instance in Figure 2, we have a feature group formed by the pairs  $(o_3, o_2)$ ,  $(o_3, o_1)$  in the time-intervals  $[\tau_1, \tau_2] \cup [\tau_2, \tau_3]$  and  $[\tau_5, \tau_6] \cup [\tau_6, \tau_7]$ . Indeed, considered  $\epsilon_1$ =“away” ( $\epsilon_i$  for  $F^1$ ),  $\epsilon_{\min_2}=25$ ,  $\epsilon_{\max_2}=50$  (respectively,  $\epsilon_{\min_i}$  and  $\epsilon_{\max_i}$  for  $F^2$ ), the values of the feature  $F^1$  are the same (“away”) and the values of the feature  $F^2$  have the same numeric range. These conditions hold in the feature time-intervals  $[\tau_1, \tau_2] \cup [\tau_2, \tau_3]$  and  $[\tau_5, \tau_6] \cup [\tau_6, \tau_7]$ , but they do not hold in the time-intervals  $[\tau_3, \tau_4] \cup [\tau_4, \tau_5]$  because the value of the feature  $F^1$  is “close” which is different from “away”.

**DEFINITION 3 (COMMUNITY).** A set of feature groups  $\{\mathcal{G}_{f_1}, \mathcal{G}_{f_2}, \dots, \mathcal{G}_{f_n}\}$  defines a community  $\mathcal{C}$  iff:

- the feature groups  $\mathcal{G}_{f_1}, \mathcal{G}_{f_2}, \dots, \mathcal{G}_{f_n}$  consists of the same pair group  $\mathcal{G}=\mathcal{G}_1=\mathcal{G}_2 = \dots = \mathcal{G}_n$  composed by  $(m-1)$  pairs of objects with the same reference object and the same set of  $m-1$  participants.
- the feature time-intervals of two different feature groups are disjoint  $([\tau_i, \tau_{i+k}] \cup [\tau_{i+k+1}, \tau_{i+2k}] \cap [\tau_p, \tau_{p+k}] \cup [\tau_{p+k+1}, \tau_{p+2k}]) = \emptyset$  and chronologically ordered ( $i + 2k < p$ ).

The sequence of the feature time-intervals associated with the feature groups is called time-line.

For instance, in Figure 2, we have a community formed by the pairs  $(o_3, o_2)$  and  $(o_3, o_1)$  in the time-line  $[\tau_1, \tau_3]$ ,  $[\tau_3, \tau_5]$ ,  $[\tau_5, \tau_7]$ , where  $o_3$  is the reference object,  $o_2$  and  $o_1$  are participant objects. In particular, in the feature time-intervals  $[\tau_1, \tau_2] \cup [\tau_2, \tau_3]$  and  $[\tau_5, \tau_6] \cup [\tau_6, \tau_7]$ , the feature  $F^1$  has value “away”, while the feature  $F^2$  has values in the range  $[25, 50]$  ( $\epsilon_1$ =“away”,  $\epsilon_{\min_2}=25$ ,  $\epsilon_{\max_2}=50$ ). In the feature time-interval  $[\tau_3, \tau_4] \cup [\tau_4, \tau_5]$ , the feature  $F^1$  has value “close”, while the feature  $F^2$  has values in the range  $[15, 25]$  ( $\epsilon_1$ =“close”,  $\epsilon_{\min_2}=15$ ,  $\epsilon_{\max_2}=25$ ).

To capture possible discontinuities, we should handle the case in which  $i + 2k < p - 1$ , namely when the feature time-intervals are separated over time. At this aim, we introduce an input parameter  $\gamma$  which defines the maximum temporal gap that can be admitted between two feature time-intervals.

Now, we can give a formal statement of the problem of discovering communities from trajectories:

**Given** a set of moving objects  $\mathcal{O}$  and the corresponding trajectories, a set of time-stamps  $\mathcal{T}$ , the features  $\mathcal{F}$  and the width of the associated time-intervals  $\Delta$ , **Discover** the communities as formalized in Definition 3: for each community  $\mathcal{C}$ , the temporal gap in the time-line does not exceed  $\gamma$  and the number of involved objects is greater than or equal to the minimum input threshold  $\min\mathcal{O}$ .

### 3. PROPOSED METHOD

The proposed solution comprises three steps: *i*) transformation of the original trajectories in descriptive spatio-temporal features, *ii*) arrangement of the feature vectors produced in the previous step in a tree-like structure in order to generate feature groups and *iii*) discovery of communities from feature groups.

#### 3.1 Transformation of Trajectory Data

Tracking devices often record the positions of moving objects with irregularity and discontinuity, mainly due to physical and instrumental factors which can affect the data quality. To remove possible inconsistencies we have to handle this kind of error sources. Moreover, the analysis of interactions among objects, we intend to conduct, suggests that we should apply a pre-processing step able to return positions (of the objects) equally distanced over time, so that the trajectories can be handled with regular timing. We adopt a data transformation technique which first performs a temporal segmentation operation and then projects the segmented trajectories into the descriptive space. Preliminarily, an outlier removal operation is applied on the trajectories.

The temporal segmentation performs a discretization step on the set  $\mathcal{T}$  and generates time-intervals  $[\tau_i, \tau_{i+k}]$ ,  $[\tau_{i+k+1}, \tau_{i+2k}]$ ,  $\dots$ ,  $[\tau_p, \tau_{p+k}]$ ,  $[\tau_{p+k+1}, \tau_{p+2k}]$  with width equal to  $\Delta$ . This allows to have a sort of re-sampling of the trajectories at regular time-stamps. In particular, for each object a single geo-spatial location is associated with the set of positions observed in each time-interval (segment). This location is determined by an aggregation operation applied to the original positions in a time-interval. As aggregation operator we prefer to use the geometric mean due to its simplicity and because other pre-processing operations (such as, smoothing and interpolation) could introduce data loss and potential creation of artifact in the trajectory data.

The descriptive space includes spatio-temporal features defined to model the interactions and changes of the movements of pairs of objects. The use of new descriptors to represent the original trajectories is not novel. In the literature we can find several types of features (also called movement parameters) which have been defined basically for eliciting information which the trajectories are not able to do directly [3]. Typically, features are produced by simple feature extraction algorithms applied to original trajectories and their purpose is to model physical and spatial characteristics of the movements, such as speed, acceleration, duration, direction, etc. In this work, the features are extracted from the aggregate values computed in two consecutive time-intervals (segments). More precisely, the value of a feature is computed for each pair of objects and it is determined from the two aggregate values computed in the respective time-intervals for each object of the pair. We investigate six features defined as follows (please refer to Figure 3):

*Categoric Reciprocal Movement (CRM)* is the feature which represents the movement of an object with respect to the



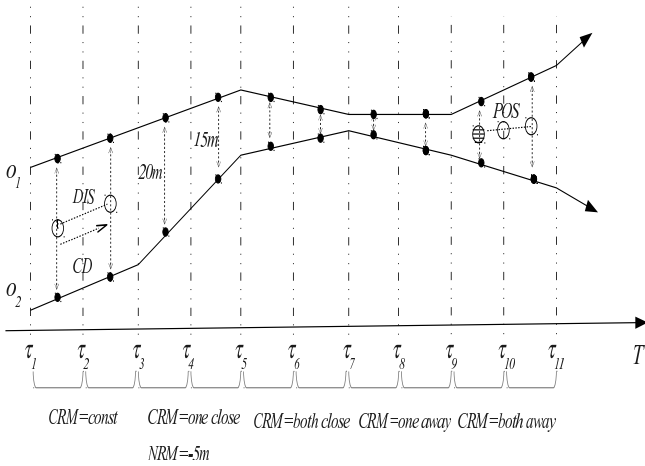


Figure 3: Trajectory transformation.

movement of another one. It takes five possible categoric values in function of the two aggregate locations. The set of possible values was defined manually and comprises {"one\_away", "both\_away", "const", "one\_close", "both\_close"}. More specifically, "const" corresponds to two objects that travel together by keeping their distance constant. We have "one\_away" when one of the two objects is moving away from the other one while the latter does not change. The value "one\_close" occurs when one of the two objects is moving close while the trajectory of the other one remain unchanged. The value "both\_away" corresponds to two objects that are moving away from each other. On the other hand, when the trajectories tend to move close we have "both\_close".

*Numeric Reciprocal Movement (NRM)* is the feature which, like CRM, represents the movement of an object relatively to another one but with numeric values. The value of NRM is derived from the distances computed, in each time-interval, between the two aggregate locations of the pair of objects. It is equal to the difference between these two distances. Thus, when two objects are moving close to each other, NRM has a negative value, while, otherwise the value is positive.

*Displacement (DIS)* denotes the displacement done by the pair of objects over the two time-intervals. The value of DIS is derived from the middle locations between the two aggregate locations (in the two time-intervals) and it is equal to the distance between the two middle locations.

*Cardinal Direction (CD)*. The features listed above provide a spatial description of the movement without specifying any geographic connotation. We introduce the feature CD in order to elicit the information about the spatial direction and capture that information as the classical cardinal direction of the movement of the pair of objects. The value of CD is derived from the middle locations between the two aggregate locations (in the two time-intervals) and it takes the direction which goes from the middle location of the first time-interval towards the middle locations of the second time-interval.

*Position (POS)*. The purpose of this feature is to provide information on the localization of the movement. Indeed, the features listed above cannot distinguish whether two identical movements are localized in the neighbourhood or in completely distant locations. The value of POS is derived from the middle locations between the two aggregate locations (in the two time-intervals) and it corresponds to

the middle point of the two middle locations.

Finally, for each pair of objects  $o_u$  and  $o_v$ , the transformation technique returns a vector of valued features  $\langle \text{CRM}, \text{NRM}, \text{DIS}, \text{CD}, \text{POS} \rangle$  computed on the two consecutive time-intervals  $[\tau_i, \tau_{i+k}]$ ,  $[\tau_{i+k+1}, \tau_{i+2k}]$ . It is worthwhile that the extraction of features for each pair of objects on consecutive time-intervals has a two-fold result: *i*) modelling the interaction of the smallest admissible group of objects (namely, two objects), and *ii*) capturing relevant changes of their movements which turn out to be evident only on time-intervals rather than instantaneous time-stamps.

### 3.2 The Feature Tree

Once the feature vectors have been generated, they populate a B-tree [2] which is used to discover first the feature groups and then the communities. The tree structure does not change when the vectors are inserted and it is defined on the basis of the set of features introduced in Section 3.1. The arrangement of tree levels is such to represent the features in the order  $\{\text{CRM}, \text{NRM}, \text{CD}, \text{DIS}, \text{POS-x}, \text{POS-y}\}$  (Figure 4(a)). The feature ordering is decided by domain experts on the basis of their criteria about the discriminative power of the features. Thus, the features CRM and NRM are ranked first because they depict, better than the others, the interaction in a pair of objects. Then, we place features CD and DIS because they are able to denote characteristics on the changes of the movement, and, finally the features POS-x and POS-y which provide a spatial indication not directly related to the interactions and changes in moving objects.

Nodes of a specific level refer to one feature and access to nodes (children) of the lower level which, in their turn, refer to another feature. More precisely, a node has as many child nodes as the number of the possible values of the feature associated with its level, therefore, the number of nodes of a specific level is equal to the number of the possible values of the feature associated with the parent level. In the case of categorical features, the child nodes are denoted with distinct values  $\epsilon_l$  defined in Section 3.1. For example, at the level associated with feature CD, the nodes have eight child nodes, one for each value of the set {"north", "north-east", "east", "south-east", "south", "west", "south-west", "north-west"}. In the case of numeric features, the child nodes are denoted with distinct ranges  $[\epsilon_{min_l}, \epsilon_{max_l}]$  produced by a discretization technique. In this work, we adopt equifrequency discretization since it guarantees the balancing of the tree due to the uniform distribution of vectors to ranges.

This tree structure allows us to collocate in the same node the vectors whose values of the feature are identical ( $\epsilon_l$ ) or are included in the same range ( $[\epsilon_{min_l}, \epsilon_{max_l}]$ ). The root node contains vectors which have only one feature with identical value (categoric), while the leaf nodes contain vectors which have all categoric features with identical values and all numeric features with values included in the same ranges. Therefore, the vectors collocated in the same leaf node will be those that have traversed the same path in the tree and that we consider similar since share the same categoric values and same numeric ranges. For instance, in the leftmost leaf node in Figure 4(b), the pairs  $(o_2, o_1)$ ,  $(o_2, o_3)$  share the same categoric values, namely "one away" for CRM and "north" for CD, and the same numeric ranges, namely [1,3] for NRM, [20,40] for DIS, [100,600] for POS-x, and [50,150] for POS-y.

The insertion process starts at the root and descends the tree. For each level, it chooses the node whose value of the

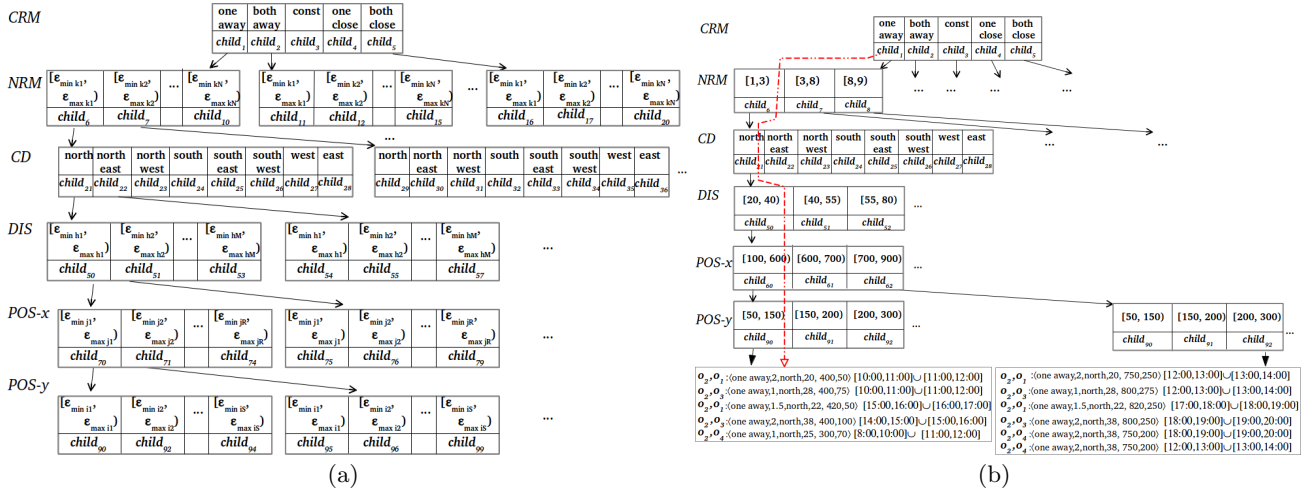


Figure 4: Feature tree example. The red dotted line illustrates a path.

associated feature is identical to (categorical) or includes (numeric) the value of the same feature of the current vector. From the chosen node we access to its child nodes where we replicate the insertion considering the appropriate feature until to reach the leaf nodes.

### 3.3 Feature Groups and Communities

We exploit the structure of the feature tree to determine the geo-spatial and temporal components of feature groups which are, in their turn, necessary for the communities. From each leaf node we can extract at least one feature group. The pair group  $\mathcal{G}$  of a feature group can be searched among the pairs of the inserted vectors, while the geo-spatial component is determined directly from the tree path which characterizes each leaf node. The temporal component is computed by the method given in the sequel.

The method analyses the content of the leaf nodes separately and, for each of these, it searches the feature time-intervals which are in common. In particular, the method identifies all possible pair groups (Definition 1) present in each leaf node and, for each pair group, it processes all sequences of feature time-intervals in order to find the sequences in common. The analysis is thus focused on each pair group and is conducted in two phases: first, generation of candidate sequences, then, selection of the more interesting candidates with respect to preference criteria.

In the first phase, we adopt the efficient algorithm proposed in [6] in order to find sequences (candidates) in common between a reference sequence of feature time-intervals and the set of all sequences of the current pair group. The algorithm solves the problem by searching the intersections between the feature time-intervals of the reference sequence and the feature time-intervals of the remaining sequences. In particular, for each time-interval of the reference sequence, the algorithm uses two binary search operations, one into the sorted list of the time-stamps which terminate the time-intervals and the other into the sorted list of the time-stamps which open the time-intervals. Each search excludes the time-intervals that cannot intersect the query interval. A detailed description can be found in [6]. Eventually, the intersecting time-intervals are sorted and combined to form the candidate sequences.

In the second phase, two selection operations are per-

formed, one subsequent to the other one. The first one filters out the candidates which have time-intervals shorter than the width  $\Delta$ , while the second one selects the candidates which meet the preference criteria. We have two preference criteria, one alternative to the other one. The first criteria (*maxInterval*, *MI*) favor feature groups with time-intervals as long as possible, whereas the second criteria (*maxObjects*, *MO*) favor time-intervals associated with large set of pairs. The preference criterion is strictly connected to the choice of the reference sequence seen in the first phase. When we choose *maxInterval* the reference sequence is chosen as the longest sequence in the set of all sequences of the pair group, while when we choose *maxObjects* the reference sequence is chosen as the shortest sequence which has the maximum number of pairs, since feature groups with higher number of pairs are more probable in shorter sequences.

The result of the two phases consists in only one sequence which contains the feature time-intervals shared in the current pair group. It provides a temporal characterization which completes the description of the feature group.

According to Definition 2 and the structure of the feature tree, a reference object is associated with only one feature group in each leaf node. Thus, a reference object is associated with a set of feature groups  $\{\mathcal{G}_{f_1}, \mathcal{G}_{f_2}, \dots, \mathcal{G}_{f_n}\}$  computed from all leaf nodes. These feature groups anyway have different sets of participant objects. The method for discovering communities follows this same idea and builds groups of moving objects relatively to reference objects. It works with the feature groups of the same reference object and operates on the selected sequences of the feature time-intervals by generating a sequence of ordered feature time-intervals (time-line) with the same set of participant objects.

Two alternatives are adopted depending on the chosen preference criterion (*maxObjects* or *maxInterval*). They operate in the same way (Algorithm 1) but they differ in that the first one aims at generating time-lines with large number of participant objects, whereas the second one aims at generating time-lines with the long feature time-intervals. Both variants start by sorting (in chronological order) the sequence of the time-intervals of the feature groups associated with the current reference object  $o_r$ . This may return in an ordering where the time-intervals of the same feature group are separated and time-intervals of different feature

---

**Algorithm 1** COM /\* community discovery \*/

---

**Input:**  $\{\mathcal{G}_{f_1}, \mathcal{G}_{f_2}, \dots, \mathcal{G}_{f_n}\}, \gamma, o_r, minO$   
**Output:**  $\mathcal{T}_{lines}$

- 1:  $\mathcal{T}_L \leftarrow \emptyset; \mathcal{T}_{lines} \leftarrow \emptyset; \mathcal{D} \leftarrow \emptyset;$
- 2:  $S \leftarrow \text{sort\_by\_time}(\{\mathcal{G}_{f_1}, \mathcal{G}_{f_2}, \dots, \mathcal{G}_{f_n}\});$
- 3:  $F_{prev} \leftarrow \text{nextTimeInterval}(S, \tau_1);$
- 4:  $\text{insert}(\mathcal{T}_L, F_{prev}); \text{remove}(S, F_{prev});$
- 5: **while**  $S \neq \emptyset$
- 6:    $F_{next} \leftarrow \text{nextTimeInterval}(S, \text{getEndTimeStamp}(F_{prev}));$
- 7:   **if**  $\text{gap}(F_{prev}, F_{next}) \leq \gamma$
- 8:     **if**  $\text{testParticipants}(\text{getParticipants}(F_{prev}),$   
           $\text{getParticipants}(F_{next}))$
- 9:        $\text{update}(\mathcal{T}_L, F_{next}); \text{remove}(S, F_{next})$   
       $F_{prev} \leftarrow F_{next};$
- 10:    **else**
- 11:       $\text{insert}(\mathcal{D}, F_{next}); \text{remove}(S, F_{next});$
- 12:    **else**
- 13:       $S \leftarrow S \cup \mathcal{D}; \mathcal{T}_L \leftarrow \emptyset;$
- 14:       $\mathcal{D} \leftarrow \emptyset; \mathcal{T}_{lines} \leftarrow \mathcal{T}_{lines} \cup \{\mathcal{T}_L\};$
- 15:       $F_{prev} \leftarrow \text{nextTimeInterval}(S, \tau_1);$
- 16:       $\text{insert}(\mathcal{T}_L, F_{prev}); \text{remove}(S, F_{prev});$
- 17: **prune\\_by\\_minO}(\mathcal{T}\_{lines});**

---

groups are adjacent.

Algorithm 1 generates a time-line incrementally by testing joining the next time-interval ( $\text{getNextTimeInterval}()$ ) to the current time-line ( $\mathcal{T}_L$ ). In particular, the time-interval  $F_{prev}$  is considered for the join when *i*) it follows temporally the last time-interval added to the time-line  $\mathcal{T}_L$  and there is no time-interval with the same participants which precedes it, and *ii*) it is not temporally distant more than  $\gamma$ . The application of the test distinguishes two techniques: for *maxObjects* the test is implemented as  $\text{getParticipants}(F_{prev}) = \text{getParticipants}(F_{next})$ , while for *maxInterval* the test is  $\text{getParticipants}(\mathcal{T}_L) \cap \text{getParticipants}(F_{next}) \neq \emptyset$ , where  $\text{getParticipants}(\mathcal{T}_L)$  returns the participants which are in common to the time-intervals added to  $\mathcal{T}_L$ . The output ( $\mathcal{T}_{lines}$ ) is a set of candidate time-lines which are further processed: the time-lines with number of participants less than  $minO$  are pruned, then, from those remaining, we select only the time-line which better satisfies the preference criterion (either highest number of participants or longest sequence of time-intervals).

**EXAMPLE 2.** We extract feature groups and communities based on Figure 4(b) for  $\Delta=1$  hour. On the leftmost leaf node, we have a feature group  $\mathcal{G}_{f_1}$  whose pair group is composed by the pairs  $(o_2, o_1)$ ,  $(o_2, o_3)$ , the geo-spatial component is equal to “one away” (CRM), “north” (CD), [1,3] (NRM), [20,40] (DIS), [100,600] (POS-x), and [50,150] (POS-y), while the temporal component corresponds to the sequence of intersecting feature time-intervals  $\langle [10:00,12:00], [15:00,16:00] \rangle$ . On the rightmost leaf node, we see a feature group  $\mathcal{G}_{f_2}$  whose group consists of the pairs  $(o_2, o_1)$ ,  $(o_2, o_3)$ , and  $(o_2, o_4)$  the geo-spatial component is equal to “one away” (CRM), “north” (CD), [1,3] (NRM), [20,40] (DIS), [700,900] (POS-x), and [200,300] (POS-y), while the temporal component corresponds to the sequence of intersecting feature time-intervals  $\langle [12:00,14:00], [18:00,19:00] \rangle$ . Let  $o_2$  be the reference object and  $\gamma=4$  hours. The time-intervals are sorted as follows  $\langle [10:00, 12:00] (\mathcal{G}_{f_1}), [12:00, 14:00] (\mathcal{G}_{f_2}), [15:00, 16:00] (\mathcal{G}_{f_1}), [18:00, 19:00] (\mathcal{G}_{f_2}) \rangle$ . By choosing the criterion *maxObjects*, we obtain the community composed of the pairs  $(o_2, o_1), (o_2, o_3)$ , and  $(o_2, o_4)$  which exhibit on the time-line  $\langle [12:00, 14:00] [18:00, 19:00] \rangle$  the movement so described: “one away” (CRM), “north” (CD), [1, 3] (NRM), [20, 40] (DIS), [700, 900] (POS-x), and [200, 300] (POS-y). Instead, by choosing the criterion *maxInterval*,

we obtain the community composed by the the pairs  $(o_2, o_1)$  and  $(o_2, o_3)$  which exhibit the movement “one away” (CRM), “north” (CD), [1, 3] (NRM), [20, 40] (DIS), [100, 600] (POS-x), [50, 150] (POS-y) in [10:00, 12:00], [15:00, 16:00], and the movement “one away” (CRM), “north” (CD), [1,3] (NRM), [20, 40] (DIS), [700, 900] (POS-x), [200, 300] (POS-y) in [12:00, 14:00], [18:00, 19:00]. ■

## 4. PERFORMANCE EVALUATION

Experiments were conducted in order to test the efficiency of **COM** and the influence of the parameters on the discovered communities with both preference criteria (**COM-MO** and **COM-MI**). Also, we performed comparative experiments with two competitors. The first one (**TC**), is used as baseline and it aims at discovering the common sub-trajectories with a density-based line-segment clustering algorithm [7]. It takes as user-defined parameters the minimum number of line-segments and radius of the clusters. The second one (**SW**), discovers groups of objects moving for certain snapshots that could be not consecutive [8]. The algorithm **SW** works on pre-existing clusters and adopts a candidate generation strategy. It takes as user-defined parameters the minimum number of the objects and minimum duration the swarms (which correspond to  $minO$  and  $\Delta$  of **COM**). We note that **SW** cannot be directly applied, since it does not handle either trajectories of different length or missing values. To perform a fair comparison, we tested it on the pre-processed trajectories returned by the temporal segmentation (Section 3.1).

We evaluated the performance of the algorithms using two real-world datasets: *i*) **Microsoft Geolife**<sup>1</sup> comprising trajectories of 182 users outdoor movements in a period of over three years sampled every 1-5 seconds or every 5-10 meters. This dataset contains almost 24 millions of observations in a set of 18 millions time-stamps. *ii*) **Starkey**<sup>2</sup> which has been generated by the Starkey project and contains radio-telemetry locations of the movements of 128 elks. The observation period is May 1993-August 1996 and comprises 168,000 distinct recordings in 166,000 time-stamps. Each object has a portion of 0.09 observations per time-stamp. In both datasets, trajectories have different length and can contain positions recorded at different time-stamps.

Figures 5(a) and 5(b) illustrate the results of the efficiency when tuning  $minO$ . The results of **SW-AVG** include also the running times averaged on  $\Delta=\{1/2, 1, 1.5, 2\}$  hours, while those of **TC** are averaged on several settings of the input parameters. We observe that the running times of **COM** are significantly lower than those of **SW** and **TC** (*y*-axis is logarithmic). In addition, the performance of **COM** with respect to **SW** can be explained with the fact that **SW** spends time in a preliminary density-based clustering and exploration of the search space of the candidate swarms. **TC** requires more time because the clustering decision requires a distance measure on sub-trajectories whose execution is computationally intensive. **COM** exhibits the best runtimes also when tuning  $\Delta$  (Figures 5(c) and (d)) but with a different behaviour due to the different density of the trajectories, as said before: in Geolife we have essentially a slight decreasing tendency, while it is increasing in Starkey. The decrease exhibited by **SW**, when increasing  $\Delta$ , is due to the reduced number of clusters that are likely to be extracted from wider time-intervals.

<sup>1</sup><http://research.microsoft.com/apps/catalog/>

<sup>2</sup><http://www.fs.fed.us/pnw/starkey/data/tables>

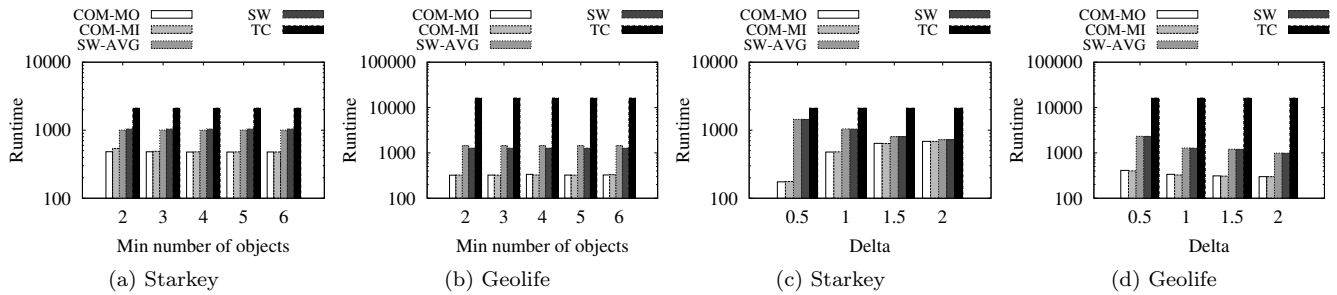


Figure 5: Runtime (in seconds) vs.  $\min O$  and  $\Delta$  ( $\gamma=1$  hour).

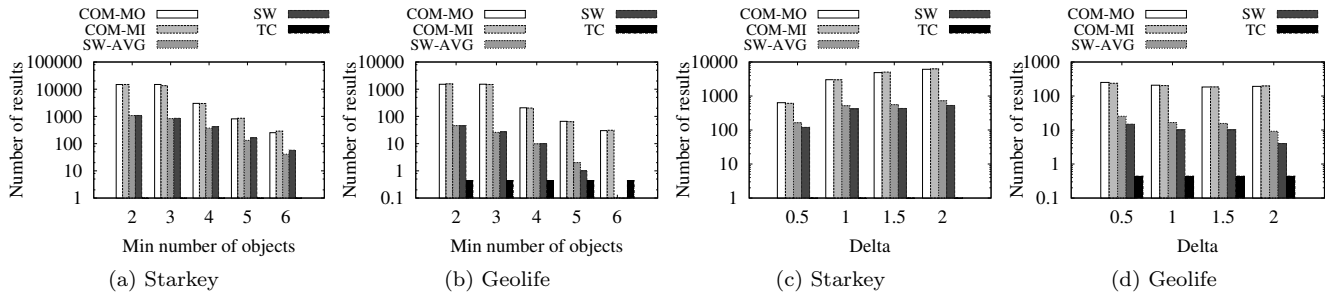


Figure 6: Number of results vs.  $\min O$  and  $\Delta$  ( $\gamma=1$  hour).

The different density and distribution of the two datasets is the key to analyze the number of the groups (communities, swarms and clusters) when varying  $\Delta$  (Figure 6,  $\min O=4$ ). The results of **SW-AVG** are averaged on  $\min O=\{2,3,4,5,6\}$ . **COM** and **SW** have similar behaviour, namely slightly decreasing in Geolife and increasing in Starkey. This comforts us about the response of our approach with respect to trajectories which have very different characteristics. A deeper inspection reveals the different order of magnitude between the communities and swarms: this is quite expected since **SW** works on the spatial closeness of the objects, while **COM** can generate groups of objects even when they are not close. Instead, **TC** discovers an average number of clusters which is less than one. This difficulty could be due to the inherent complexity that an operation of grouping of line-segments can raise with respect to grouping simple geo-spatial locations, as in the case of **SW**.

## 5. CONCLUSIONS

We investigated the problem of mining groups of moving objects from trajectory data. Different from the existing approaches relying on the spatial closeness, our work considers the interactions among the objects and changes of their motions which opens to the possibility of following the complete dynamics of a group. The proposed solution integrates an efficient grouping technique which avoids to re-scan all data. Experiments remark the efficiency with respect to other algorithms. We plan to extend the work in several directions including: *i*) the integration of pre-processing techniques (e.g., locality sensitive hashing) to guide the discovery process on sets of moving objects of particular interest, *ii*) the adaptation of the approach to a distributed architecture (e.g., MapReduce framework) to analyze massive trajectory data, and *iii*) the construction of the feature tree without considering any pre-defined order of the features.

**Acknowledgments:** In partial fulfilment of PRIN 2009 Project “Learning Techniques in Relational Domains and Their Applications” funded by the Italian Ministry of University and Research.

## 6. REFERENCES

- [1] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Comput. Geom.*, 41(3):111–125, 2008.
- [2] D. Comer. The ubiquitous b-tree. *ACM Computing Surveys*, 11:121–137, 1979.
- [3] S. Dodge, P. Laube, and R. Weibel. Movement similarity assessment using symbolic representation of trajectories. *International Journal of Geographical Information Science*, 26(9):1563–1588, 2012.
- [4] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.
- [5] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, pages 364–381, 2005.
- [6] R. M. Layer, K. Skadron, G. Robins, I. M. Hall, and A. R. Quinlan. Binary interval search: a scalable algorithm for counting interval intersections. *Bioinformatics*, 29(1):1–7, 2013.
- [7] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *ACM SIGMOD Conference*, pages 593–604, 2007.
- [8] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.
- [9] Z. Li, J. Han, M. Ji, L. A. Tang, Y. Yu, B. Ding, J.-G. Lee, and R. Kays. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM TIST*, 2(4):37, 2011.
- [10] S. Liu, S. Wang, K. Jayarajah, A. Misra, and R. Krishnan. Todmis: mining communities from trajectories. In *CIKM*, pages 2109–2118, 2013.
- [11] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *GIS*, pages 99–108, 2010.
- [12] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *Data Eng. Bull.*, 33(2), 2010.

# Mobile Sensing Data for Urban Mobility Analysis: A Case Study in Preprocessing

Indrė Žliobaitė, Jaakko Hollmén  
Helsinki Institute for Information Technology HIIT  
Aalto University School of Science, Department of Information and  
Computer Science, PO Box 15400, FI-00076 Aalto, Espoo, Finland  
indre.zliobaite@aalto.fi, jaakko.hollmen@aalto.fi

## ABSTRACT

Pervasiveness of mobile phones and the fact that the phones have sensors make them ideal as personal sensors. Smart phones are equipped with a wide range of motion, location and environment sensors, that allow us to analyze, model and predict mobility in urban areas. Raw sensory data is being collected as time-stamped sequences of records, and this data needs to be preprocessed and aggregated before any predictive modeling can be done. This paper presents a case study in preprocessing such data, collected by one person over six months period. Our goal with this exploratory pilot study is to discuss data aggregation challenges from machine learning point of view, and identify relevant directions for future research in preprocessing mobile sensing data for human mobility analysis.

## Categories and Subject Descriptors

H.2.8 [Database Applications]: Data Mining; I.5.2 [Pattern Recognition]: Feature evaluation and selection

## Keywords

mobile sensing, data preprocessing, feature extraction, accelerometer, smart cities

## 1. INTRODUCTION

The availability and penetration of smart mobile devices is increasing; smartphone penetration in Europe is already more than 49% [2]. Mobile sensing systems are finding their way in many application areas, such as monitoring human behavior, social interactions, commerce, health monitoring, traffic monitoring, and environmental monitoring [9].

Pervasiveness of mobile phones and the fact that they are equipped with many sensor modalities makes them ideal sensing devices. Since the mobile phones are personal devices, we can use the idea of mobile sensing to probe the owner of the phone and the environment, in which the user is moving. Our general interest is to use mobile phones to

learn about the mobility patterns of people and to reason and predict about their mobility patterns in urban traffic environment.

The idea of using mobile phones as sensors is not new: mobile phones have been used for context recognition (e.g. [8]) and for measuring social interactions (e.g. [4]) in complex social systems already about a decade ago.

Nowadays, smart phones are equipped with a wide range of sensors, including motion, location and environment sensors, that allow collecting rich observational data about human mobility in urban areas. Various predictive modeling tasks can be formulated based on such data. For example, one can be interested in recognizing the current activity of a person [11], predict next location [6], or predicting a trajectory of movement [13]. In this study, we explore challenges of preprocessing such sensory data for machine learning purposes for analyzing, modeling and predicting human mobility in urban areas. We present an experimental case study, report lessons learned and discuss challenges for future research.

The task of data preprocessing in mobile sensing is not trivial, and there are various challenges associated with this task. Data from sensors is collected as a sequence of time stamped observation records. Data records are not equally time spaced. Moreover, the timestamps of records from different sensors are not matching. In addition, observation records can be of different types: recording discrete events (e.g. battery charger plugged in), continuous processes (e.g. acceleration), or static measurements (e.g. current temperature).

The standard machine learning approaches for predictive modeling require data to be represented as instances. Instance (or example, case, or record) is defined as a single object of the world from which a model will be learned, or on which a model will be used (e.g., for prediction) [10]. However, data recorded by mobile sensors does not come as instances. Data comes as time stamped records, where time stamps are different for each sensor and are not equally spaced in time. The main data preprocessing question is, how to aggregate such data and convert it into instances for machine learning.

The problem of sensory data preprocessing is also not new, typically in the literature an arbitrary data aggregation approach is chosen and briefly mentioned (or not reported at all). However, there is a lack of dedicated studies focusing on the problem of preprocessing itself. Furthermore, the existing literature on preprocessing of mobile sensing data mainly deals with feature extraction from one sensor (e.g.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

**Table 1: Data collection rates (in sec.)**

Source	Until June 14		After June 14	
	Per.	Dur.	Per.	Dur.
AccelerometerSensorProbe	-	-	30	30
ActivityProb	-	-	30	30
AndroidInfoProbe	86 400	-	86 400	-
BatteryProbe	1 800	-	1 800	-
BluetoothProbe	-	-	300	-
CellProbe	600	-	300	30
GravitySensorProbe	-	30	30	30
GyroscopeSensorProbe	120	-	30	-
HardwareInfoProbe	86 400	-	86 400	-
LightSensorProbe	1 800	30	120	30
LocationProbe	1 800	30	120	30
MagneticFieldSensorProbe	1 800	30	120	30
OrientationSensorProbe	120	30	30	30
RotationVectorSensorProbe	-	-	30	30
RunningApplicationsProbe	60	-	-	-
TemperatureSensorProbe	1 800	-	30	-
WifiProbe	600	-	300	-
DataUploadPeriod	3 600	-	3 600	-
Data annotation	manual		manual	

accelerometer or GPS signal) [5, 14], which is only one side of the problem. We are not aware of research works dealing with the problem of synchronizing data from multiple sensors.

This pilot study reports an exploratory case study in aggregating mobile sensing data and can be seen as the first step towards systematic treatment of this problem. We investigate how to construct instances out of sensory data for analyzing human mobility in three different scenarios: aggregating data with manual event annotations, converting static records to estimates of dynamic characteristics, and aggregating data from multiple sensors for predictive modeling purposes. The main contribution of this study is identification (via case study), and discussion of data aggregation challenges, as well as highlighting important questions for future research. To demonstrate the nature of mobile sensing data, we have released a data set called *Sensing Venice*, which is available as an open data set at the authors’ websites.

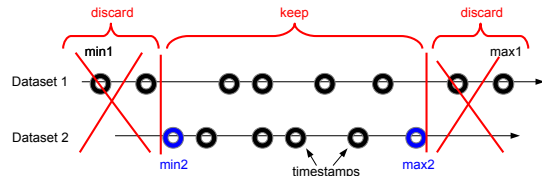
The paper is organized as follows. Section 2 presents a case study consisting of three experiments in analyzing mobility data. Section 3 summarizes the main challenges and lessons learned from the case study with respect to data preparations, and presents a taxonomy of settings and guidelines for data aggregation. Section 4 discusses open research directions and concludes the study.

## 2. EXPERIMENTAL CASE STUDY

We start the case study with a description of the settings and data, and then report methodologies and results from the three experiments. The experiments were selected to capture different nature of data aggregation challenges in processing mobile sensing records. The focus of this study is on data preparation techniques, therefore, we do not go further into domain driven analysis of the outcomes of the case studies and their implications from traffic management point of view. Domain-dependent treatise is left for future work.

### 2.1 Data collection and representation

Dataset has been collected using contextLogger3 [12], which



**Figure 1: Alignment of the time ranges.**

is an open source software tool for smartphone data collection and annotation based on Funf open-sensing framework [3]. Data was collected during the period from 2013 February 7 to 2014 January 27 using Sony Ericsson Xperia Active phone, which is using Android OS, v2.3 (Gingerbread). Summarizing our data collection upto January 2014, we have over 300 million timestamped records, resulting in approximately 13 Gb of data. The capacity of the battery is 1200 mAh.

On June 14 the settings for data collection rates were changed. Table 1 provides details on sampling period and duration. Period indicates how often a given sensor is activated, and duration indicates for how long the sensor is activated. For example, if the period is 120 and the duration is 30 it means that the accelerometer is activated every 120 sec, and is collecting data for 30 sec.

We have released a mobile sensing dataset — coined *Sensing Venice* — as open data collection, which is available at <http://users.ics.aalto.fi/jhollmen/Data/>. This particular subset of data has been recorded in July 2013 in Venice, Italy. We hope that the mobile sensing dataset demonstrates the nature of the original raw data, and encourages other researchers to develop solutions to the challenges introduced in the current paper.

The experiments reported in this paper are based on the full six months dataset.

### 2.2 Experiment 1: processing event annotations

The first experiment investigates processing of event annotations. The start and the end time of an event is input by a user. These event annotations need to be cleaned, pre-processed and aligned with the recorded sensor data.

We illustrate these data preprocessing challenges by an experiment in modeling accelerometer data collection rate for different user activities. Accelerometer data is available only from June 14, hence, we use only that period of data in this experiment.

#### 2.2.1 Methodology

We have two sets of recordings: event annotations and accelerometer records. Both are timestamped, but the timestamps are not aligned in any way. First we find the minimum (earliest) and the maximum (latest) time stamps in both sets, and discard the records from non-overlapping parts, as illustrated in Figure 1.

The main challenge in data preparation in this experiment is to extract activity labels from the event annotations. Annotations provide the start and the end time stamps of activities. Starts and ends are not necessarily paired, i.e., it may happen that there is a start, but no end, or there are three starts in a row and then one end of the same activity.

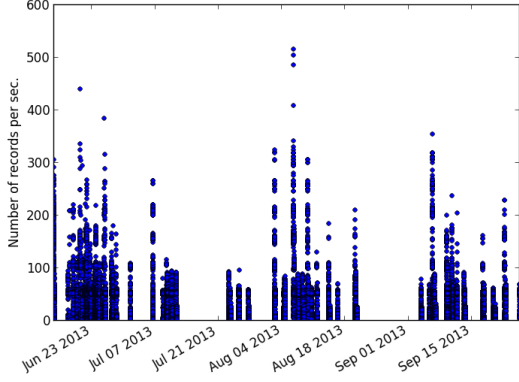


Figure 2: Accelerometer records over time.

We process annotations in a sequence. If there is a start, we consider an activity happening (no matter how many other starts of the same activity follow) until either of the following three triggers appear: annotation "stop", annotation "invalidate", or more than 6 hours have passed since the start. The latter rule is arbitrary chosen, assuming that mobility activities are typically short time.

For every second in time we create a label vector, where currently ongoing activities are encoded as 1, and not ongoing activities are encoded as 0. We get a label matrix  $\mathbf{A}$  of size  $T \times k$ , where  $T$  is the number of seconds from the beginning of data recording to the end, and  $k$  is the number of distinct activities recorded. Obviously, longer recording periods produce very large data files, therefore, one may consider choosing a larger time step for aggregation (e.g. creating a vector for every 10 sec. instead).

For modeling data collection rates, we need to process automatically collected accelerometer data and align it with the extracted activity labels. The time step, over which data is aggregated, needs to match the step used for label extraction earlier. We count the number of accelerometer records per second for every second that accelerometer was on. We get a vector  $\mathbf{X}$  of size  $T \times 1$ , where each entry is a number of records per second. Figure 2 shows the amount of data recorded over time.

Given the extracted label matrix  $\mathbf{A}$  and the record vector  $\mathbf{X}$ , we can obtain estimates for average records per second for each activity. There is an important modeling decision to be made. If two or more activities take place at the same time, how does it affect the number of records? Suppose activity  $a_1$  generates  $n_1$  records per second, and activity  $a_2$  generates  $n_2$  records. We could assume that if  $a_1$  and  $a_2$  take place at the same time,  $n_1 + n_2$  records are generated. Alternatively, we can assume that if  $a_1$  and  $a_2$  take place at the same time,  $\max(n_1, n_2)$  records are generated. In our experimental study we take the latter approach.

Following the first assumption, data collection rate can be modeled as a linear regression, where the inputs are binary indicators of activities, and the output is the number of records generated. If the second assumption is adopted,

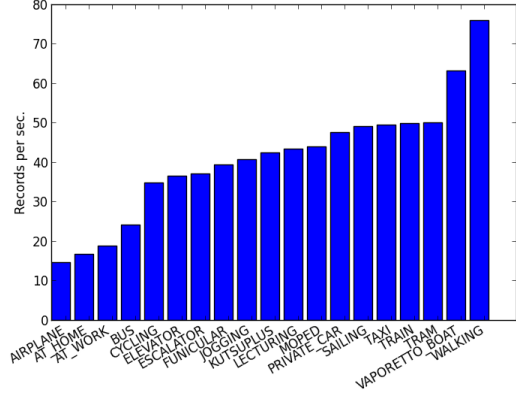


Figure 3: Average number of accelerometer records per second for each activity.

each activity is modeled independently, as follows:

$$\bar{r}_i = \frac{\sum_{j=1}^T a_{ji} x_j}{\sum_{j=1}^T a_{ji}}, \quad (1)$$

where  $i$  denotes the  $i^{th}$  activity,  $a_{ji}$  is the  $j^{th}$  entry of activity  $i$  in matrix  $\mathbf{A}$ ,  $x_j$  is the  $j^{th}$  entry of vector  $\mathbf{X}$ .

Note, that this approach will automatically exclude the periods when the phone was off and no data was collected, since in those cases  $x_j = 0$ .

With this experimental setup we anticipate that different activities generate different number of accelerometer records. Raw sensor data in Android is acquired by monitoring sensor events. A sensor event occurs every time a sensor detects a change in the parameters it is measuring [1]. We expect different activities to have different acceleration patterns, and in turn to result in different data collection rates.

### 2.2.2 Results and observations

Figure 3 shows the resulting estimates of data collection rates for each activity. Data aggregated in such a way can be used, for instance, as a feature for activity recognition. While this feature stand alone would not be enough to separate all the activities, certain activities could be well distinguished, for instance, walking.

We see that walking produces the most records per time period, while at home or in the office activities produce the least. These results intuitively make sense. At home or office the phone would typically stay still on the table, hence, there is not much motion involved.

Moreover, we can see that conceptually similar activities appear close together, presenting similar amount of records. For example, "elevator" is very close to "escalator" and "funicular", where we would expect a smooth not too fast movement following a straight path. On the other spectrum of the scale "train" and "tram" appear nearby, both are means of transportation over rail. From this pilot experiment we can conclude that this preprocessing approach works and proceed to the next experiment.



## 2.3 Experiment 2: estimating the rate of change from static measurements

Sensors record static measurements; however, sometimes our interest may be to estimate dynamic characteristics. Examples include estimating speed of a moving object from GPS coordinates, estimating energy consumption from battery level indications, estimating flow rates from observed level of liquid.

The task in this experiment is to estimate how much energy is being consumed during data collection, given unequally time spaced observations of the battery level. The main challenges are: deriving conversion equations, filtering out uninformative observations, identifying and handling the periods with missing information (when the data collection application is off).

### 2.3.1 Methodology

For energy rate estimation we use level, voltage and status information from the BatteryProbe. Level indicates the percentage of battery charge remaining. Voltage indicates current voltage. Status indicates whether the phone is charging, discharging or the battery is full. All the records have the same time stamps.

Energy consumption in watt-hours (Wh) is computed as

$$E_{(Wh)} = Q_{(mAh)} \times V_{(V)} / 1000, \quad (2)$$

where  $Q$  is the electric charge in milliampere-hours (mAh),  $V$  is voltage in volts (V).

Given data recorded by ContextLogger2, the electric charge during the  $i^{th}$  time period, which starts at time  $t_i$  and ends at time  $t_{i+1}$  can be estimated as

$$Q_i = Q_{battery} \times (L_i - L_{i+1}), \quad (3)$$

where  $L_i$  and  $L_{i+1}$  are battery levels (in percentage) at the start and the end of the period.

However, there are two challenges. Firstly, data records are not equally spaced in time. As a result, time period  $i$  is not necessarily equal to  $i + 1$  and, hence,  $Q_i$  is not comparable to  $Q_{i+1}$ . Secondly, battery levels are presented in low granularity (in rounded percents). As a result, estimation becomes stepped, where for several records the estimated energy consumption is zero (because  $L_i = L_{i+1}$ ), then suddenly jumps and becomes zero again.

The first challenge can be overcome by estimating the rate of energy consumption instead of the amount of energy consumed. The rate of consumption is known as power  $P$  (in Watts), which during time period  $i$  can be computed as

$$P_i = Q_i \times 3600 / (t_{i+1} - t_i). \quad (4)$$

It is assumed that  $t$  is measured in seconds.

The second challenge can be addressed by discarding all the records of battery level, where the level remains the same as in the preceding record. This way we get less time intervals to consider, while the intervals themselves are longer.

### 2.3.2 Results and observations

Figure 4 plots the resulting energy consumption rate over time. We can see that most of the time energy consumption with ContextLogger is around 5 W. Negative energy appears when the phone is plugged for charging.

There are higher peaks of energy power, which may be due to switching ContextLogger on and off, when it is partially charged. In order to estimate energy more exactly at these



Figure 4: Estimated energy power estimates over time.

points, we would need to know or detect when context logger is switched on and off. Currently this information is not available from the logs.

Overall, from this pilot experiment we can conclude that it is possible to estimate the distribution of dynamic characteristics, such as energy consumption, from static sensor observations. However, this kind of preprocessing requires some domain knowledge input (e.g. knowing from physics how energy is defined). Nevertheless, we anticipate that it is possible to define a generic model form of such estimates for any sensor. This remains a subject of future investigation.

## 2.4 Experiment 3: data aggregation for predictive modeling

The goal of this experiment is to model energy consumption as a function of charging status of the battery.

### 2.4.1 Methodology

We model energy consumption as a linear function of indicator variables of the charging status: "discharging", "charging" and "full". We assume that energy consumption or inflow should be fully covered by these three sources; hence, we do not model the intercept (assume that the intercept is zero). With preliminary experiments using cross-validation we chose the Ridge regression optimization approach [7] ( $\lambda = 1$ ) for finding the regression coefficients.

In the first experiment we discarded the observations, which did not indicate any change in the battery level. In this experiment we use all the records. We select an aggregation step  $s$  (e.g. 1 hour), which will be used to form data instances from raw observation records.

Energy consumption data is produced as specified in Algorithm 1. Voltage is estimated as  $(V_i + V_{i+1})/2$ . Energy power is estimated as in Eq. (4). We first divide all the time span into time periods of length  $s$ . Within each period we find all the observed records. We calculate energy consumption from record to record over time. Finally, we normalize the energy consumed from the actually observed time period to a fixed size time period  $s$ .

For example, if our period of aggregation is one day (24h), we may not necessarily observe records from 00:01 to 23:59.

It may happen, that we observe records only from 8:00 to 18:00. In such case, the factual time period is 10 hours. Hence, we would divide the observed energy consumption by 10 and multiply by 24 (the actual period of interest).

---

**Algorithm 1:** Aggregation of energy consumption data.

---

**Data:** A time ordered sequence of battery level  $L$ , voltage  $V$ , and timestamps  $t$  ( $N$  records); battery capacity  $Q_{battery}$ ; aggregation step  $s$  (in sec)

**Result:** Dataset: energy consumption  $\mathbf{E} = (E_1, E_2, \dots)$  (Wh) during time period  $s$

```

1 for  $b = 1$  to  $(t_N - t_1)/s$  // number of bins
2 do
3    $E_b \leftarrow 0, T_b \leftarrow 0;$ 
4   for  $t_{now} \in [t_1 + (b-1)s, t_1 + bs - 1]$  do
5     // all time stamps within an interval
6      $E_b \leftarrow E_b + Q_{battery}(L_{now} - L_{now+1}) \times$ 
7      $\times (V_{now} + V_{now+1})/2000;$ 
8      $T_b \leftarrow T_b + t_{now+1} - t_{now};$ 
9   end
10  $E_b \leftarrow sE_b/T_b;$ 
11 end

```

---

Charging status data is aggregated in a similar way, as energy. For each time period  $b$  we have a three-dimensional vector of battery status, where each dimension indicates the percentage of time spent "discharging", "charging" or operating with "full" battery. The final dataset is a matrix with four columns, where the first three columns are the indicators of battery status, and the last column is the energy consumption. Each row corresponds to an observation period of 1 hour.

Since different sensors and sampling rates were active in the first and in the second period of data collection (before June 14 and after), we run the experiment in two parts, corresponding to these periods. For each period data is split into training and testing at random (50:50%). The regression parameters were estimated on the training part, and the model was tested on the testing part.

### 2.4.2 Results and observations

Table 2 presents the predictive models and their respective accuracies. The coefficients at the charging status mean the estimated energy consumption per hour. For example, 0.41 discharging means that when data is being collected and the phone is not plugged in, it consumes 0.41Wh of energy per hour. The negative coefficients mean that this is the net amount of energy the phone *gets*, when it is plugged into the electricity source.

We see that the directions of energy consumption (positive or negative) are identified correctly in both cases. In the second period discharging when a charger is plugged is excessively high (0.76), identifying reasons for that requires

**Table 2: Energy consumption as a function of battery status.**

Period	Discharg.	Charg.	Full	MAE	$R^2$
Until June 14	0.41	-0.89	0.23	0.15	54%
After June 14	0.27	-1.66	0.76	0.19	72%

further investigation. The relative magnitudes of energy consumption in the first period are convincing: charging is faster than discharging (0.89 Wh vs 0.41 Wh), and discharging when the charger is plugged is slower than when no charger is plugged (0.23 vs. 0.41 Wh). Interestingly, the energy consumption estimate is lower after June 14. It could be because of more inactivity periods during the second span; however, a further investigation is needed to analyze this phenomenon. Moreover, battery level is estimated rounded numbers, therefore the resulting energy consumption estimate is stepped and approximate.

Overall, from this experiment we conclude that it is possible to uncover, model and interpret relationships between processes with basic data aggregation; however, more investigation into accompanying data denoising is required, which remains a subject of future investigation.

## 3. DISCUSSION

The three case studies illustrate different challenges with data preprocessing. The first challenge is aggregating unevenly spaced and not synchronized in time observations, observed in Experiment 1. Given two sequences of observations, first we discard non overlapping (in time) parts, and then aggregate data over a fixed time step (1 sec).

Setting an appropriate aggregation time step presents one challenge for future investigation. The smaller the step, the faster the reaction time. However, the accuracy of the analysis may suffer if the step is too small to present an informative summary of what is happening. On the other hand, an excessively large time step only slows down the reaction time (e.g. a person starts walking, but recognition is delayed). Moreover, a large time step may capture heterogeneous data, for example, a mixture of several activities.

Another important open challenge is how to distinguish the periods of inactivity from the periods when no data is being collected, observed in Experiments 1, 2, 3. In this study we assumed that if there are no accelerometer records, then there is no activity. This is a crude approximation. Accelerometer sensor may be off or accelerometer sampling rate may be set to very large value (e.g. sample every 10 min). Failing to distinguish periods of inactivity from the periods when no data is being collected introduces noise in the resulting computational models. Such noise could be ignored, if there were only a few periods of inactivity or no data collection. However, when analyzing human mobility typically there are many more inactive periods than active periods. Unless a person is, for instance, a taxi driver, during a typical working day there would be several spans of movement and quite a lot of inactive periods, when the phone is resting in a bag or on a table. Therefore, reliable methods for filtering out the periods of no data collection and disambiguating the periods of inactivity need to be developed.

The next open challenge is how to deal with different granularity of sensor records, as observed in Experiment 3. For example, battery level is estimated in percentage, there are no decimals of percentage. If we are estimating energy consumption, the battery level would remain constant for a while before changing. If does not mean; however, that during that period no battery has been consumed. In Experiment 2 we overcame this challenge by introducing a variable data aggregation step, which varies depending on observed changes in the battery level. However, to deal with this challenge systematically we need some kind of smoothing

mechanism, that would also work online.

Finally, automatically processing manual annotations presents a big open challenge. Ideally, manual annotation of an activity should have a start and an end. In practice, an activity may have, for instance, multiple starts and no end, or an end, but no start. In addition, some activity time stamps may have manual corrections. In such a case end may happen earlier than the start, as we observed in Experiment 1. One way to deal with this challenge could be just to discard such corrupted data. However, manually annotated data is typically very scarce, therefore it is in the best interest of analysis to preserve as much of it as possible. Therefore, tailored data cleaning and imputation methods are needed. In our experimental investigation we introduced several simple intuition based rules to check and correct the integrity of user annotations. For example, if an activity starts and "end" annotation does not arrive for 6 hours, we consider the activity finished. A systematic generic approach to this problem is needed, that is a subject of future investigation.

## 4. SUMMARY AND CONCLUSIONS

We investigated how to aggregate mobile sensing data for machine learning purposes. We performed three exploratory experiments to illustrate different data preprocessing challenges. Following the experimental study, we identified and discussed several major challenges in mobile sensing data preprocessing for urban mobility analysis. The main directions are: how to determine the aggregation step, how to identify and isolate the periods of inactivity, how to deal with different granularity of observations, how to effectively automatically process manual data annotations, and integrate them with the observational data. To accompany the paper, we have released a subset of the data as openly available data, coined *Sensing Venice*. The data with its documentation is available at the authors' websites.

This pilot study sets a basis for further investigation aiming at producing a systematic methodology for preprocessing mobile sensing records for predictive modeling.

## 5. ACKNOWLEDGMENTS

This work was supported by the Aalto University AEF research programme and Academy of Finland grant 118653 (ALGODAN).

## 6. REFERENCES

- [1] Android developer's guide. [http://developer.android.com/guide/topics/sensors/sensors\\_overview.html](http://developer.android.com/guide/topics/sensors/sensors_overview.html).
- [2] Mobile economy europe 2013. Report, GSMA, 2013.
- [3] AHARONY, N., PAN, W., IP, C., KHAYAL, I., AND PENTLAND, A. Social fMRI: Investigating and shaping

social mechanisms in the real world. *Pervasive and Mobile Computing* 7, 6 (2011), 643–659.

- [4] EAGLE, N., AND PENTLAND, A. Reality mining: sensing complex social systems. *Personal and Ubiquitous Computing* 10 (2006), 255–268.
- [5] FIGO, D., DINIZ, P. C., FERREIRA, D. R., AND CARDOSO, J. A. M. Preprocessing techniques for context recognition from accelerometer data. *Personal Ubiquitous Comput.* 14, 7 (2010), 645–662.
- [6] GAO, H., TANG, J., AND LIU, H. Mobile location prediction in spatio-temporal context. In *The Proceedings of Mobile Data Challenge by Nokia Workshop at the 10th Int. Conf. on Pervasive Computing* (2012).
- [7] HASTIE, T., TIBSHIRANI, R., AND FRIEDMAN, J. *The Elements of Statistical Learning*. Springer New York Inc., 2001.
- [8] HIMBERG, J., KORPIAHO, K., MANNILA, H., TIKANMÄKI, J., AND TOIVONEN, H. Time series segmentation for context recognition in mobile devices. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001)* (2001), pp. 203–210.
- [9] KHAN, W. Z., XIANG, Y., AALSALEM, M. Y., AND ARSHAD, Q.-A. Mobile phone sensing systems: A survey. *IEEE Communications Surveys and Tutorials* 15, 1 (2013), 402–427.
- [10] KOHAVI, R., AND PROVOST, F. Glossary of terms. editorial for the special issue on applications of machine learning and the knowledge discovery process. *Machine Learning* 30, 2/3 (1998).
- [11] KWAPISZ, J. R., WEISS, G. M., AND MOORE, S. A. Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.* 12, 2 (2011), 74–82.
- [12] MANNONEN, P., KARHU, K., AND HEISKALA, M. An approach for understanding personal mobile ecosystem in everyday context. In *Effective, Agile and Trusted eServices Co-Creation — Proceedings of the 15th International Conference on Electronic Commerce ICEC 2013* (2013), vol. 19 of *TUCS Lecture Notes*, pp. 135–146.
- [13] MONREALE, A., PINELLI, F., TRASARTI, R., AND GIANNOTTI, F. Wherenext: A location predictor on trajectory pattern mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2009), KDD, pp. 637–646.
- [14] ZHANG, J., XU, J., AND LIAO, S. S. Aggregating and sampling methods for processing gps data streams for traffic state estimation. *IEEE Transactions on Intelligent Transportation Systems* 14, 4 (2013), 1629–1641.

# Crowd Density Estimation for Public Transport Vehicles

Marcus Handte,  
Muhammad Umer Iqbal,  
Stephan Wagner,  
Wolfgang Apolinarski,  
Pedro José Marrón  
NES  
University of Duisburg-Essen  
first.last@uni-due.de

Eva Maria Muñoz  
Navarro,  
Santiago Martínez  
Investigación y Desarrollo  
ETRA  
(emunoz|smartinez).etra-  
id@grupoetra.com

Sara Izquierdo  
Barthelemy,  
Mario González  
Fernández  
Proyectos Europeos  
EMT de Madrid  
first.last@emtMadrid.es

## ABSTRACT

Existing information systems for urban public transportation are empowering travelers to optimize their trips with respect to travel duration. Experience with such systems shows that this is a viable approach. However, we argue that solely relying on trip duration as the primary indicator for satisfaction can be limiting. Especially, in urban settings providing additional information such as the expected number of passengers can be highly beneficial since it enables travelers to further optimize their comfort. As technical basis for determining the number of passengers, we have built an inexpensive hard- and software system to estimate the current number of passengers in a vehicle. Furthermore, we have deployed the system in several buses in the city of Madrid. In this paper, we describe the overall design rationale, the resulting system architecture as well as the underlying algorithms. Furthermore, we provide an initial report on the system's performance. The initial results indicate that the system can indeed provide a reasonable estimate without requiring any manual intervention.

## Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous

## General Terms

WLAN Monitoring, Presence Detection, Intelligent Transport Systems, Smart Cities

## 1. INTRODUCTION

Today, most information systems for urban public transportation are empowering travelers to optimize their trips with respect to travel duration. To do this, they integrate static information about routes and schedules with dynamic information about unexpected delays. On top of this they

provide planning engines that compute shortest paths in order to minimize the trip duration for the travelers.

Clearly, past experiences with such systems shows that this is a viable approach that is useful for many travelers. However, we argue that solely relying on trip duration as the primary indicator for traveler satisfaction can be limiting as it hides many other facets that impact the travelers comfort. Examples may include environmental information such as the accessibility of different vehicles for travelers with special needs or dynamic information such as the likelihood of being able to get a seat in a particular vehicle.

Especially, in urban settings where the same destination can be reached over multiple routes or where the same route is traversed by different vehicles frequently, providing additional information can be highly beneficial. For example, considering the former case, a traveler might simply be able to slightly adjust his route whereas in the latter case, a traveler might simply have to start a trip earlier or later in order to improve his or her level of comfort.

Besides from trip duration, a main influential factor for the overall level of satisfaction with a particular public transport option is the overall crowdedness of the vehicles. However, in the absence of a mandatory reservation system or a fine-grained trip-based payment system, capturing the number of passengers is a challenging and costly task that is typically done by means of manual counting. Yet, in order to provide real-time information on a city-scale such manual approaches are clearly ill-suited.

In this paper, we describe an alternative approach to determine the number of passengers in a vehicle. Based on this approach, we have built an inexpensive hard- and software system to estimate the current number of travelers in a vehicle. Furthermore, we have deployed the system in several buses in the city of Madrid. In addition to the estimation of number of travelers, our system also estimates the location of buses between the bus stops. Based on this deployment, we provide an initial report on the system performance. The results indicate that the system can indeed provide a reasonable estimate for the number of passengers inside the vehicle as well as a reasonable estimate of the location of buses between two stops.

The remainder is structured as follows. In the next section, we briefly discuss the underlying design rationale. Thereafter, in Section 3, we outline the overall approach. In Section 4, we describe details of our implementation and in Section 5, we report initial results of our deployment in the city of Madrid. In Section 6, we discuss related work and finally,

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

in Section 7 we conclude the paper with a short summary and an outlook on future work.

## 2. DESIGN RATIONALE

As described previously, our goal is to provide a system to determine the number of passengers in a particular vehicle of a public transportation system in order to provide the resulting crowd density information to the travelers. As a result of this overarching goal, we can derive the following five sub goals:

- *Sufficient accuracy:* To provide meaningful information, the system should be able to determine the number of passengers accurately. Thereby, it is important to note that given the typical capacity of vehicles the system does not have to be perfect. Instead, smaller deviations can be tolerated as long as the overall tendency of the crowd density reflects the real situation.
- *Full automation:* To be reliable and feasible to deploy, the system should not rely on manual intervention by passengers. Furthermore, it should not put additional stress on the support personnel such as the driver or the guards. Instead, the system should be able to determine the number of passengers automatically.
- *Low cost:* To be scalable to a city level, the hardware cost of the system should be minimal. As a result, the system should only consist of low-cost off-the-shelf components and it should optimally leverage the existing infrastructure.
- *Low latency:* To provide fresh information to the travelers, the system should be able to report changing numbers of passengers quickly such that it can not only be used for advance planing based on historical data but also to support ad hoc decisions by travelers based on the current state.
- *Low privacy impact:* To be acceptable for the passengers of the public transport system, the system should be non-intrusive from a privacy perspective. Furthermore, it should only gather information that is needed to provide the service and ideally, it should be hard to retrofit the information for non-related use cases.

## 3. APPROACH

Based on the five goals, we describe our overall approach in the following. To do this, we first describe the basic idea and the resulting system architecture. Thereafter, we describe the details of the algorithms used for crowd density estimation and vehicle tracking. In the next section, we describe the implementation details for our deployment in several buses in the city of Madrid.

### 3.1 Overview and Architecture

Our approach for estimating the number of passengers in a vehicle can be considered a specialized variant of the smart phone tracking approach described in [6]. The basic idea is that WLAN-enabled mobile devices are periodically sending so-called *probe requests* as part of their IEEE802.11 protocol operation to detect the access points that are present in their surroundings. In order to completely cover the frequency spectrum during their scans, the devices typically

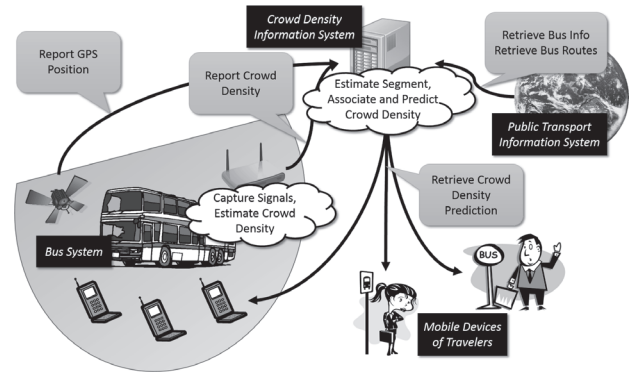


Figure 1: System Architecture

repeat their probe request on all available channels. Thus, given adequate network monitoring hardware, it is possible to overhear these request by simply tuning into one of them. Moreover by continuously monitoring the presence and absence of the probe requests, it is possible to accurately count the mobile devices that are in the vicinity of the network monitoring hardware.

Once the number of passengers has been estimated, it needs to be made accessible to the travelers. To do this, it is first transmitted to a central server where it is then combined with the associated segment of the current route of the vehicle. To compute this association, we rely on the positioning information provided by the vehicle itself by means of a built-in GPS receiver. We then combine with the static route information managed by the public transport operator with the GPS position to determine the current route segment that the bus is traversing. As a last step, we then store the vehicles route segment with the associated crowd-level and a timestamp. Finally, the resulting data is made accessible to travelers which can then retrieve the crowd density estimations for the public transportation system for different times of day through their mobile devices.

The overall system architecture is depicted in Figure 1. It consists of three main components, namely the system inside the vehicle which is responsible for determining the crowd density and capturing the current GPS position, the public transport information system which is responsible for providing geo-spatial information about the routes that the vehicles are operating on as well as a crowd density information system which integrates the information and makes it accessible to travelers. While there are many possible options to split up the responsibility of determining crowd density from WLAN signals, we decided to keep all computations regarding probe requests local to the system inside the vehicle. This means that apart from GPS position, the system solely transfers the current crowd density. The reason for this is twofold. First, this reduces the overall bandwidth requirements when compared to transferring all probe requests to the server. Second, it also protects the privacy of the passengers since the transferred data is hard (and in most cases impossible) to attribute to individual passengers.

In the following, we describe the two main issues, namely the crowd density estimation in the vehicle as well as the vehicle tracking at the server-side in more detail.



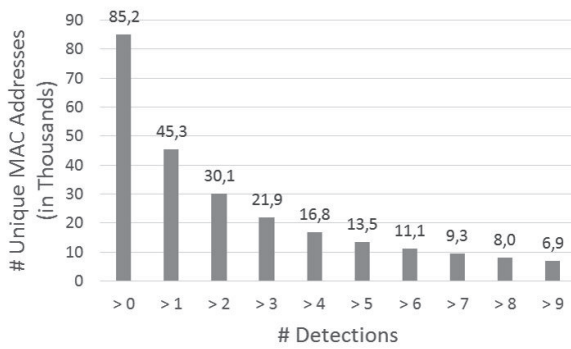


Figure 2: Detected Devices over 14 Day Period

### 3.2 Crowd Density Estimation

As indicated previously, our approach to crowd density estimation is based on the idea that WLAN-enabled devices are periodically sending *probe requests* in order to detect the access points that are nearby. In order to completely cover the frequency spectrum during probing, the devices typically repeat their probe request on all available channels. Using a WLAN device that is put into monitoring mode, it is possible to receive the probe requests of nearby devices by simply monitoring a particular channel. By keeping track of the MAC addresses of the devices sending out the probe requests, it is then possible to determine the time duration that a certain device is close to the monitoring device.

When applied to public transportation, an important difference between prior work and our scenario is that in our case, the monitoring WLAN device is a) mobile – since it is mounted inside a vehicle – and b) often moving through a densely populated area. As a consequence, we can expect that the monitoring device will not only receive signals from mobile devices that are located in the vehicle but it will also receive signals from devices that are simply nearby the vehicle. This problem is amplified by the fact that in typical public transportation networks, stops at important locations (e.g. in the city center) are targeted by multiple lines. Thus, when a vehicle is stopping in order to allow passengers to enter and exit the vehicle, passengers waiting for another vehicle from another line will be detected as well.

To demonstrate this problem and to develop a solution for it, we have installed a WLAN monitor in one bus operating in the city of Madrid, Spain during a period of 14 days. During the time, the bus was operated for 224 (out of 336) hours and while it was operating, we logged the probe requests received by the monitor. To avoid duplicate detections of the same requests sent out multiple times, we limited the amount of logged probe requests to 1 request per MAC address per second.

In total, the monitor logged 384874 probe requests from 85212 unique MAC addresses. However, as indicated in Figure 2, from these unique MAC addresses approximately 40000 were only seen once and an additional 15000 addresses were only seen twice. These numbers clearly demonstrate the fact that a significant fraction of mobile devices were most likely not traveling in the bus. Instead, it is more likely that they were located at a crowded bus stop or somewhere close to the street where the bus was driving.

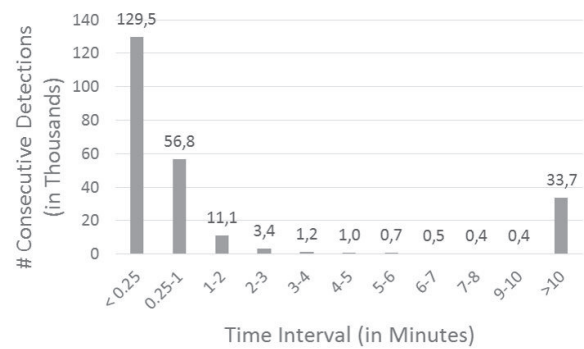


Figure 3: Probe Request Interval Distribution

To filter out these MAC addresses, while still being able to report changes quickly, we decided to integrate a sliding window mechanism that would remove addresses that were not detected over a longer period of time. In order to configure the windowing period, we further analyzed the logs to determine the typical rate at which we would detect probe requests from devices.

Figure 3 shows the results extracted from the logs. As indicated, the vast majority of probe requests - approximately 185000 - are transmitted within one minute. From these requests, roughly 125000 are transferred within 15 seconds or less, meaning that they are most likely repeated requests that were not filtered out by our 1s rate limitation. The remaining 60000 requests, however, are sent at least 15 seconds later which indicates that they might be new requests. Looking at the overall slope indicated by the histogram in Figure 3, it seems apparent that the vast majority of consecutive probe requests are heard typically within 1 and at most within 3 minutes. Interestingly, the histogram also shows that there is a significant number of consecutive probe requests that are repeated within an time frame above 10 minutes. However, we attribute these to stationary devices that are picked up multiple times during the 14 day period when the bus traverses routes multiple times.

Given these results, we configure our sliding window mechanism for the crowd density estimation to 3 minutes. In order to avoid the counting of devices that are not within the bus, we suppress devices that have not been detected for at least 1 minute and we continue to count them until their signals are no longer contained in the window - meaning that the WLAN monitor has not received a probe request for at least 3 minutes.

### 3.3 Vehicle Tracking

Once the crowd density has been estimated, it needs to be assigned to a particular route and segment (i.e. the pair of previous stop and next stop of the vehicle). However, in European cities, estimating the route that a vehicle is taking by simply connecting the different stops will result in a very coarse grained estimate of the route. Instead, it is necessary to model the route by means of a more detailed representation such as a polygonal path that defines multiple waypoints between the stops.

To determine the current location of the vehicle using the possibly imprecise GPS, we rely on basic geometric opera-

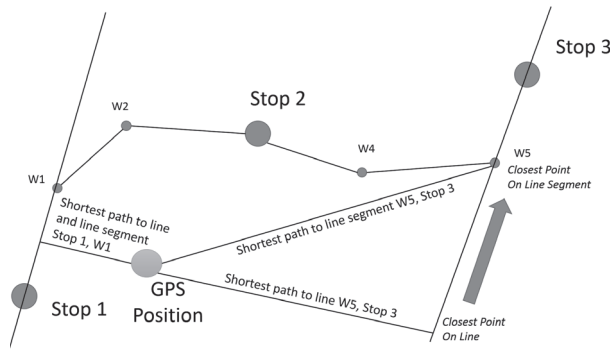


Figure 4: Vehicle Tracking Approach

tions on top of an accurately modeled polygonal paths representing the routes. Thereby, the basic idea is to compute the shortest paths to all line segments as depicted in Figure 4. Technically, this is done in three steps. First, we compute the closest point to each line segment of the path. Note that this is either the perpendicular line between the line segment and the GPS position (left) or in cases where the perpendicular line does not intersect within the segment, it is one of the two points defining the line segment (right). Then, we compute the distance between the GPS position and the closest point for all line segments and finally, we use the segment with the shortest distance as the current position on the route which identifies the previous and the next bus stop.

To minimize the computational overhead of the resulting computations in a spherical coordinate system, we simply interpret the GPS coordinates as Cartesian coordinates. While this may result in imprecisions when applied to larger distances, we did not find this problematic at a city level. To test this, we tracked three buses over the course of 2 weeks and verified the validity of the resulting bus stop sequences by comparing them with the route information. In all cases, the bus stop sequences were matching the sequences of the route, however, due to the limited update rate of 2 position updates per minute, some bus stops were sometimes skipped.

## 4. IMPLEMENTATION

In the following, we briefly describe a number of implementation issues that we had to tackle in order to deploy the system. To put these issues into a meaningful context, we first describe the existing infrastructure before discussing the details of our implementation.

### 4.1 Infrastructure

The Madrid bus system encompasses roughly 2000 vehicles that operate more than 200 routes. All buses are equipped with WLAN access points that provide free Internet access to the travelers. For this, the access points are equipped with a 3G network card. In addition, all buses are equipped with a GPS system. A central system polls the GPS information from the buses regularly at 30 second intervals. The gathered GPS information is then used to estimate arrival times and to dispatch new buses if delays are detected.

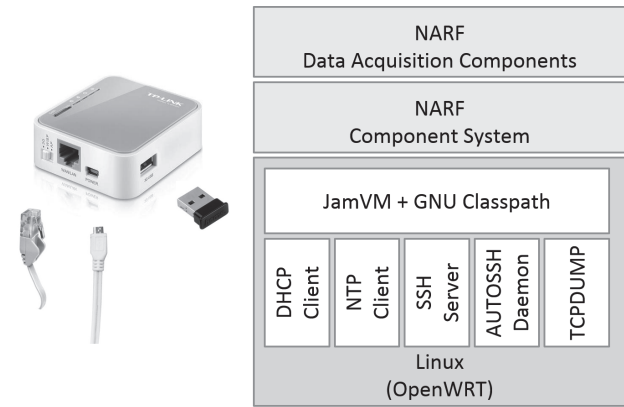


Figure 5: Bus System Hard- and Software

### 4.2 Bus System

To implement the crowd density estimation inside the buses, we rely on an additional low cost off-the-shelf access point (TP-Link 3020) as WLAN monitor which we equip with a USB memory stick to increase its internal memory for logging purposes. In order to connect the access point to the Internet, we connect it to the existing bus systems (i.e. the existing access point that provides 3G Internet connectivity to passengers). To be able to monitor the WLAN network, we replace the firmware of the device with a custom build of OpenWRT that is tailored to our needs.

Besides from packet capturing support via TCPDUMP, we install a number of system services depicted in Figure 5. To acquire an IP address from the existing access point in the bus, we run a DHCP client. In order to enable remote administration despite the firewall of the 3G network provider, we connect to one of our servers through AutoSSH and establish a tunnel to the device's SSH server. Finally, since this device does not exhibit a real-time clock, we rely on NTP in order to set its clock upon restart.

On top of this, we install JamVM with GNU Classpath in order to execute Java code. This enables us to use the NARF Component System [2] to handle the actual crowd-density measurements. To do this, we rely on existing components from the NARF component toolkit to handle the data transmission and windowing which we extend with a component that taps into TCPDUMP and interprets its output. Since our access point does not exhibit a real-time clock, we configure the device to boot up with its date set to 2012. When the NTP client on the device has successfully determined the current time at least once, this date will be adjusted to the current date (i.e. a date in 2013). In the crowd-density estimation code, we check the current time and suppress all further actions until the time is set to 2013. This effectively avoids stale readings and allows us to buffer crowd density estimations on the device together with a correct time stamp in case that the 3G connection is temporarily unavailable.

### 4.3 Public Transport Information System

To associate the crowd density information with a particular segment of a bus line, we extend the existing transport information system with 3 web services that expose some of its information. The first web service makes a list of routes available. The second service enables the retrieval of detailed



```

{
  "Id":4281,
  "LineId":17,
  "Loc":
  {
    "Lat":-0.001534102118749,
    "Lon":-7.489303515333618
  },
  "Route":33342
}

```

Figure 6: Bus Information Output Example

route information including bus stops and the polygonal line that connects them. Finally, the third service exposes the real-time information about the current bus location as well as the route that it is operating on.

All web services expose the information as JSON strings which are compact and easy to parse in most programming languages. An example for the bus information output provided by the real-time service is depicted in Figure 6. Besides from the bus id (Id) and current bus location (Loc), the output also contains the id of the bus line (LineId), which reflects the id used by the citizens and a pointer to the current route (Route) which enables the retrieval of the stops and waypoints using the route information web service.

#### 4.4 Crowd Density Information System

The last component of our implementation is the crowd density information system. Implemented as a set of Java Servlets, the system ties together the bus and route information provided by the Public Transport Information System and the crowd density estimation provided by the Bus System. To do this, it provides a web service that enables the WLAN monitor in the bus to upload its latest crowd density measurements. Furthermore, it continuously polls the Public Transport Information System in order to acquire the latest bus information.

When the Servlets are initialized or when a route change is detected, the system downloads the new route information for the bus and begins (or continues) the vehicle tracking. Whenever a new GPS coordinate for a bus is retrieved, the coordinate is matched against the polygonal path describing the route to determine the current route segment. The route segment is then associated with a timestamp and buffered in memory for future use. When a Bus System performs an upload of some crowd density information through the web service offered by the Crowd Density Information System, the system uses the timestamp that has been assigned on the Bus System when the estimation was created to determine the buffered route segment that corresponds to the reading. The resulting crowd density report for a particular route segment is then stored in a database for later retrieval through travelers.

At the present time, our implementation of the Crowd Density Information System simply provides a map-based visualization of the route information that has been captured over different time intervals. An example for this is shown in Figure 7. The black lines indicate bus routes through the city of Madrid for which crowd density information has been captured. The thickness of the lines indicate the crowd



Figure 7: Crowd Density Visualization Example

level for a particular segment of the bus route. As our next step we plan to integrate this information into a mobile bus navigation application for Android devices as part of the prototype development in the GAMBAS European FP7 research project.

## 5. EVALUATION

In the following, we evaluate our approach to crowd density detection with respect to the design goals identified in Section 2. To do this, we first discuss the system characteristics with respect to *automation*, *cost* and *privacy impact*. Thereafter, we provide an initial report on the *latency* as well as the level of *accuracy* achieved by our system.

### 5.1 Discussion

As described in Section 2, we attempt on supporting *full automation*, *low cost* while ensuring a *low privacy impact*. Given the approach and its implementation described in Section 3 and Section 4, these design goals are addressed as follows:

- *Full automation*: The presented approach for crowd density estimation is based on overhearing the probe requests that are sent by IEEE802.11 enabled mobile devices. These requests are automatically transmitted by the devices as part of their normal protocol operation. As a result, the approach will work without the installation of any additional software and thus, there is no need for passengers to be actively involved in the collection process at any point. Similarly, due to the integration with the existing services operated by the public transport provider, there is also no need for any manual intervention from drivers or other personnel. Instead, once it is installed, the complete system is fully automated.
- *Low cost*: In order to deploy our crowd density information system, we try to optimally leverage the existing infrastructure - i.e. the 3G connectivity and the GPS receiver - that is already available in the vehicles. However, in order to perform the actual monitoring we

extend the infrastructure with one additional access point. At the time of writing, the cost for the device and the USB memory stick which we are using ranges well below 50 Euros. At the server side, we introduce additional services built on top of J2EE technology. Given the platform agnostic nature of Java, they should be easy to integrate into an existing web-based infrastructure. As a result, we are convinced that the overall deployment cost of the system is reasonably low - especially, when compared to other alternatives such as camera systems, for example.

- *Low privacy impact:* Due to the fact that our system applies passive monitoring of IEEE802.11 enabled devices, it is possible to uniquely identify travelers across all vehicles of the complete public transport system. As a result, the chosen approach can be considered quite invasive from a privacy perspective. To minimize the possible negative impact on the privacy of the travelers, our implementation of the approach is distributed. Instead of collecting all raw messages at a central system, each WLAN monitor is set up to be able to compute a crowd density estimation locally. Once an estimate has been computed by the monitor, it only transmits its id, a global timestamp and the number of passengers in the bus - which is then processed and stored centrally. As a result, we argue that the privacy impact on the user is minimal. Although it may be possible to track individuals in cases where the vehicle utilization is very low (i.e. close to 1 passenger), in cases where the utilization is higher, identifying individual travelers is most likely very hard - if not impossible.

## 5.2 Experiments

To determine the degree of fulfillment with respect to the design goals of achieving a *low latency* and a *high accuracy*, we have deployed the WLAN monitors in 3 buses that are operating in the city of Madrid, Spain. At the time of writing, these buses have been collecting data for 3 weeks using the approach and implementation described in Section 3 and 4. In the following, we briefly describe our experiences with respect to latency and accuracy.

### 5.2.1 Latency

Based on the size of our windowing mechanism which uses a 3 minute window in order to determine the density of the crowd, our crowd density estimation approach introduces at least a three minute time difference. However, due to changes in network connectivity of the monitored vehicle, this latency can become temporarily higher in cases where the computed crowd density cannot be transmitted immediately. In order to visualize the probability of such cases, Figure 8 depicts the inter-reporting arrival time differences of the 75985 reports collected by our buses.

Since we configured our monitors to report crowd levels every 30 seconds (which reflects the GPS update interval of the existing transport information system), we would expect that if the vehicles 3G connection is reliable, the resulting arrival time difference would lie around 30 seconds as well. Out of the 75985 reports, 72028 reports (94.7 %) are reported with an arrival time of less than a minute and 75175 (98.9 %) are reported within 1.5 minutes or less. As a consequence, in the vast majority of all cases our crowd density

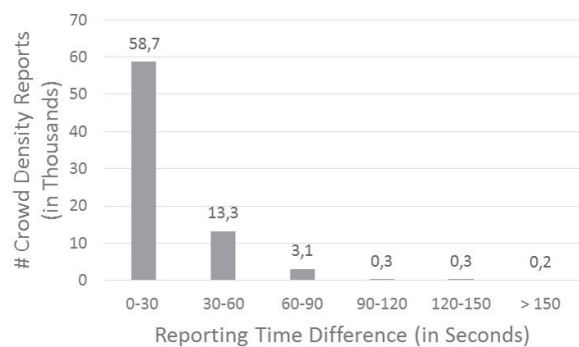


Figure 8: Crowd Density Reporting Latency

reports are available at the Crowd Density Information System within less than 5 minutes. Consequently, we think that the system is broadly applicable from a latency perspective.

### 5.2.2 Accuracy

In order to determine the accuracy of the system, we performed an initial analysis by means of manual counting the persons in one of our three buses over a 30 minutes trip from the start to the end of the bus' route. After the trip, we compared the reported crowd density measured by our system with the manually gathered information. During the experiment the bus contained between 22 and 52 passengers. Given the total capacity of 65 passengers, the bus was sometimes rather crowded. During the test, the system was able to continuously detect around 20% of the passengers on average.

To put this number in perspective, it is important to note that according to comScore, there are approximately 22.6 million smart phones in Spain<sup>1</sup> and the total Spanish population is estimated around 46.7 million persons<sup>2</sup>. Thus, we would expect that the number of persons captured by our approach would typically level off at around 49%. In addition, several smart phone users may have turned off their phone's WLAN interface in order to save power. Thus, given the rather stable 20% over trip, we believe that the approach can be used to gather reasonable crowd density estimates - however, it is clear that a more extensive study is necessary to confirm these initial results.

## 6. RELATED WORK

For a traveler two important pieces of information include when the desired vehicle is going to arrive at his/her stop and how crowded it will be. These two pieces of information pose challenges for two separate domains namely crowd density estimation and the estimation of the actual arrival time of the vehicle. For the later, this in turn requires information about the current position of the vehicle over time. In the following we give a brief overview of related work for these two domains.

<sup>1</sup>Number of smart phones in Spain available at: <http://www.comscoredata.com/2013/01/what-are-the-spanish-doing-on-their-smartphones/>

<sup>2</sup>Current estimate of the Spanish population available at: <http://en.wikipedia.org/wiki/Spain>

## 6.1 Crowd Density Estimation

Estimating crowd density in indoor and outdoor locations is an active area of research. A number of techniques has been used to estimate the crowd density with high accuracy. These techniques can be mainly classified into image processing and radio frequency based techniques. Some of the work using image processing techniques includes [8],[5],[12],[14] and [4]. [8] estimates crowd density in an outdoor environment by extracting image features using a grey level dependency matrix, minkowski fractal dimension and translation invariant orthonormal chebyshev moments. The extracted features are classified using self-organizing maps. [5] uses pixel counting approach for segmenting the foreground image from the background image and derives and proves that the geometric correction for the ground plane can be directly applied to foreground pixels. [14] provides a survey on crowd analysis techniques based computer vision and image processing. These camera based techniques though reasonably accurate requires careful mounting of cameras in buses such that maximum visual coverage is attained. Moreover, once installed further modifications of their placements is difficult to achieve and thereby is a costly and a time consuming process.

Recently crowd estimation using radio frequency based techniques have gained attention from the research community. Some of the recent work includes [11],[13], [6],[7]. [11] uses the Bluetooth transceivers on mobile phones for estimating the number of people. The approach taken by the authors is based on the assumption that considerable number of people have the Bluetooth transceiver on their mobile phones in discoverable mode. The approach relies on different information such as number of visible devices, links between visible devices, the ratio of number of devices in the current scan to the number of devices in the previous scan, device visibility durations, etc. The authors report to achieve accuracy of more than 75% in their testing scenario. [6] uses a WiFi based solution for detecting and tracking users. The system relies on detecting WiFi probes sent by mobile phones and received by WiFi monitors installed at different places. However, the WiFi probes sent by mobile phones exposes the MAC address of the device which can be used to violate user's privacy. [7] provides an insight on the vulnerability of user privacy because of exposition of such explicit identifiers. [13] uses wireless sensor network based solution for estimating crowd density. The approach employs an iterative process which includes collection and analysis of received RSSI values from the network, construction of training database using K-means algorithm and design of a spatial-temporal stability calibration mechanism to minimise noise. Apart from image processing and radio frequency based solutions there has been some work on using audio samples for estimating crowd density. [3] suggests an audio tone counting solution in which each device (mobile phone) sends a unique tone and at the same time receive tones from other devices. The sent and received tones corresponds to a bit pattern which is then combined to generate new bit pattern. The process continues until the counting is completed.

In our system presented in this paper, we have employed a radio frequency based solution. Specifically our system estimates the crowd level in the bus by keeping track of WiFi probes sent by the mobile phones of users in the bus. In this way our approach resembles with the one mentioned

in [6]. However, in contrast to that approach, our system specializes in estimating the crowd density in moving buses which requires filtering of incorrect information when the bus pass through different parts of the city. This incorrect information, in our case are the WiFi probes sent by the mobile phones in the vicinity of the bus.

## 6.2 Vehicle Tracking

In the recent years vehicle tracking has been the focus of research community. Some of the examples include [15],[1],[9],[11] and [10]. [15] presents a participatory sensing system in which users on the bus share their locations using their mobile phones with a central system which then communicate this information to other users waiting for the bus. The information is then used to predict the bus arrival time. In order to capture the user location the system relies on GSM cell tower information. For the ground truth the bus routes are divided into different segments where each end of segment is marked with three strongest GSM cell towers. The system then matches the GSM cell tower information to which the user is connected to and compare it with the ground truth to predict the location of the bus which in turn is used to predict the bus arrival time. The detection of user's presence on bus is done by detecting the audio beep generated by the ticket checking machines installed at the entrance door of the buses. [1] is a bus tracking and arrival time prediction system. The system requires smart phones to be installed on the buses. Smart phones convey the GPS coordinates of the bus and send them to a back end server. The back end server uses this information and calculates the arrival time of the bus to a particular stop and convey this information to the interested user(s). [9] is also a participatory system which require its users to install an app on their phone. The app serves two purposes, it detects whether the user is in a bus and if yes then it start sending the user's location to a back end server which then computes the arrival time for a particular stop. The detection of users presence on the bus is done by the combination of accelerometer and GPS sensors.

In our system presented in this paper the location of buses is acquired through GPS modules already installed on buses. A GPS module transmits the location of bus every 30 seconds. Our system collects this information through web services offered by the bus transportation company and using the technique described in Section 3.3 calculates the location of the bus between two stops.

## 7. CONCLUSIONS

Today, most information systems for urban public transportation are empowering travelers to optimize their trips with respect to travel duration. However, solely relying on trip duration as the primary indicator for satisfaction can be limiting. In urban settings providing more information such as the expected number of passengers can be beneficial since it enables travelers to further optimize their comfort. In this paper, we described a scalable and fully automated approach for determining the number of passengers in a vehicle. Furthermore, we discussed our experiences with a deployment of the resulting system in the city of Madrid. Our initial report on the system performance indicates that it can indeed provide a reasonable performance at low cost while preserving the travelers privacy.

At the present time, our implementation of the system

provides a rather simple map-based visualization of the route information that has been captured recently. As our next step, we are integrating the crowd information into a mobile bus navigation application for Android devices as part of the developments in the GAMBAS European FP7 research project. This application will integrate the crowd density estimations directly into the output of a trip planning engine which will enable travelers to take more informed decisions when considering the route and time of a trip. In the long run, we hope that applications like this can help to balance the load on the overall public transport system which – besides from improving the comfort of travelers – could reduce the operational costs of the network.

## Acknowledgments

This work is supported by UBICITEC e.V. (European Center for Ubiquitous Technologies and Smart Cities) and GAMBAS (Generic Adaptive Middleware for Behavior-driven Autonomous Services) funded by the European Commission under FP7 with contract FP7-2011-7-287661. The authors would like to thank the remaining members of the GAMBAS consortium for their work on and support for this paper.

## 8. REFERENCES

- [1] J. Biagioni, T. Gerlich, T. Merrifield, and J. Eriksson. Easytracker: Automatic transit tracking, mapping, and arrival time prediction using smartphones. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, pages 68–81, New York, NY, USA, 2011. ACM.
- [2] M. U. Iqbal, M. Handte, S. Wagner, W. Apolinarski, and P. J. Marron. Enabling energy-efficient context recognition with configuration folding. In *International Conference on Pervasive Computing and Communications (PerCom)*, March 2012.
- [3] P. G. Kannan, S. P. Venkatagiri, M. C. Chan, A. L. Ananda, and L.-S. Peh. Low cost crowd counting using audio tones. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 155–168, New York, NY, USA, 2012. ACM.
- [4] V. Kostakos, T. Camacho, and C. Mantero. Wireless detection of end-to-end passenger trips on public transport buses. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 1795–1800, 2010.
- [5] R. Ma, L. Li, W. Huang, and Q. Tian. On pixel count based crowd density estimation for visual surveillance. In *Cybernetics and Intelligent Systems, 2004 IEEE Conference on*, volume 1, pages 170–173 vol.1, 2004.
- [6] A. B. M. Musa and J. Eriksson. Tracking unmodified smartphones using wi-fi monitors. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, pages 281–294, New York, NY, USA, 2012. ACM.
- [7] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 user fingerprinting. In *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, MobiCom '07, pages 99–110, New York, NY, USA, 2007. ACM.
- [8] H. Rahmalan, M. Nixon, and J. Carter. On crowd density estimation for surveillance. In *Crime and Security, 2006. The Institution of Engineering and Technology Conference on*, pages 540–545, 2006.
- [9] A. Thiagarajan, J. Biagioni, T. Gerlich, and J. Eriksson. Cooperative transit tracking using smart-phones. In *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 85–98, New York, NY, USA, 2010. ACM.
- [10] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson. Vtrack: Accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 85–98, New York, NY, USA, 2009. ACM.
- [11] J. Weppner and P. Lukowicz. Bluetooth based collaborative crowd density estimation with mobile phones. In *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, pages 193–200, 2013.
- [12] J. H. Yin, S. A. Velastin, and A. C. Davies. Image processing techniques for crowd density estimation using a reference image. In *Invited Session Papers from the Second Asian Conference on Computer Vision: Recent Developments in Computer Vision*, ACCV '95, pages 489–498, London, UK, UK, 1996. Springer-Verlag.
- [13] Y. Yuan, C. Qiu, W. Xi, and J. Zhao. Crowd density estimation using wireless sensor networks. In *Mobile Ad-hoc and Sensor Networks (MSN), 2011 Seventh International Conference on*, pages 138–145, 2011.
- [14] B. Zhan, D. N. Monekosso, P. Remagnino, S. A. Velastin, and L.-Q. Xu. Crowd analysis: A survey. *Mach. Vision Appl.*, 19(5-6):345–357, Sept. 2008.
- [15] P. Zhou, Y. Zheng, and M. Li. How long to wait?: Predicting bus arrival time with mobile phone based participatory sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys '12, pages 379–392, New York, NY, USA, 2012. ACM.



# Traffic Incident Detection Using Probabilistic Topic Model

Akira Kinoshita  
The University of Tokyo  
2-1-2 Hitotsubashi, Chiyoda,  
Tokyo, Japan  
kinoshita@nii.ac.jp

Atsuhiko Takasu, Jun Adachi  
National Institute of Informatics  
2-1-2 Hitotsubashi, Chiyoda,  
Tokyo, Japan  
{takasu,adachi}@nii.ac.jp

## ABSTRACT

Traffic congestion is quite common in urban settings, and is not always caused by traffic incidents. In this paper, we propose a simple method for detecting traffic incidents by using probe-car data to compare usual and current traffic states, thereby distinguishing incidents from spontaneous congestion. First, we introduce a traffic state model based on a probabilistic topic model to describe traffic states for a variety of roads, deriving formulas for estimating the model parameters from observed data using an expectation-maximization algorithm. Next, we propose an incident detection method based on our model, which issues an alert when a car's behavior is sufficiently different from usual. We conducted an experiment with data collected on the Shuto Expressway in Tokyo over the 2011 calendar year. The results showed that our method discriminates successfully between anomalous car trajectories and the more usual, slowly moving traffic. However, our method does sometimes classify abnormally fast-moving cars as traffic incidents.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*Data mining, Spatial databases and GIS*

## General Terms

Algorithms

## Keywords

Anomaly detection, automatic incident detection, probabilistic topic model, probe-car data, traffic state estimation

## 1. INTRODUCTION

Automatic incident detection (AID) is a crucial technology in intelligent transport systems, particularly in terms of reducing congestion on freeways [10]. Traffic incidents often cause traffic congestion, causing great inconvenience and

economic loss to society. A technology that can detect traffic incidents in real time and alert people accordingly would therefore be a desirable way of reducing these ill effects.

Against this background, there have been many studies on AID, e.g., [2, 13]. Most of the approaches exploit data sent from stationary sensors and cameras installed on roads. However, the installation and maintenance of such sensors is expensive, with only the main routes likely to have them [17]. On the other hand, probe-car data (PCD), on which we focus in this paper, are becoming increasingly important, as the number of probe cars and the size of the associated data archives increase. PCD includes timestamps and the locations of vehicles, and may contain additional values such as the probe cars' speed and direction. Although a PCD system cannot monitor all cars, it enables traffic administrators to watch a vast area at a lower cost than by using stationary sensors. In addition, a PCD system can follow a probe car's sequence of movements in detail, which is hard to achieve via stationary sensors, and trajectory mining can be applied to the collected data.

Using PCD for freeways, it is easy to detect any reduction in speed, which sometimes implies congestion, by analyzing the speeds of the probe cars. However, this method is less applicable to local streets where there are many crossings and traffic lights that cause cars to stop frequently but normally. Moreover, speed reduction is not always an abnormal circumstance, even on freeways, and is not always caused by incidents such as accidents, which we would regard as sudden and unusual traffic events in this paper.

There are two types of congestion: spontaneous and abnormal [2]. Detecting spontaneous congestion is less important, as it originates in road design and urban planning. Any road may have potential bottlenecks, such as upslopes, curves, junctions, tollgates, and narrow sections. Vehicles are likely to slow down at the bottlenecks, with vehicular gaps shortening and drivers in the following cars having to brake. Congestion will then occur even without a traffic incident [7]. Spontaneous congestion also occurs when the traffic demands exceed the traffic capacity of such bottlenecks, and it is not resolved until the demand drops below the capacity [12]. The drivers may be familiar with the locations of such potential bottlenecks, and they can avoid them. On the other hand, abnormal congestion originates in traffic incidents, which need to be detected in real time to prevent or resolve any sudden heavy congestion.

In this paper, we propose an AID method for detecting traffic incidents by discovering abnormal car movements, distinguishing such movements from those occurring in spon-

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

taneous congestion. Our method measures differences between the current and usual traffic states, and has two aspects; namely, traffic state estimation and anomaly detection. First, we employ a probabilistic topic model [4] to model generation of PCD, which is influenced by hidden traffic situations, such as “smooth” and “congested.” The model introduces a single set of several hidden component states, that are associated with probabilistic distributions over the PCD values, and all the road segments have their respective mixing coefficients. Using archived PCD, maximum-likelihood parameters of the model are estimated by an expectation–maximization (EM) algorithm. The estimated model reflects the usual state over the whole observation period. Our incident detection method simply follows the intuitive meaning of “anomaly.” To detect incidents, the proposed method estimates the hidden state behind an observed PCD value and compares this current state with the usual state. If the current state is significantly different from the usual state, it is recognized as an anomaly.

We conducted an experiment using PCD observed for three of the routes of the Shuto Expressway system in Tokyo over the 2011 calendar year. The experiment showed that the proposed method can be effective for AID.

The main contributions of this paper are as follows.

- We propose a method for estimating traffic states by applying a probabilistic topic model to PCD, whereby road segments are characterized in terms of their expected performance.
- We propose a new method for detecting anomalous car trajectories according to the differences between the estimated states behind the trajectory and the usual states indicated by the learned model, whereby the detection is conducted adaptively in terms of the segments.
- Our experiment showed that the usual traffic state could be estimated using the observed PCD, and that our AID method had good selectivity for anomalous behavior by cars encountering incidents.

## 2. RELATED WORK

Although many studies have considered the traffic state estimation problem, there is no general agreement about a formal definition of a “traffic state.” Some research estimates the traffic state in terms of vehicular speed [11, 19], and this kind of estimation characterizes states, i.e., quantized speeds, as “free” or “congested” [6]. Yoon et al. [17] proposed two feature values based on vehicular speed to detect a “bad” traffic state, i.e., slow traffic. In contrast, Kerner et al. [8] used travel time. Xia et al. [15] used a clustering method to identify congested traffic in a feature space involving traffic flow, speed, and occupancy, which has been well studied in traffic engineering [12].

AID can be considered to be an application of anomaly or outlier detection. Zhu et al. [20] applied the outlier detection methods to feature vectors carefully extracted from PCD using heuristics. If an incident occurs, cars upstream of the incident will travel slower and downstream cars will travel faster. In addition, a car passing before the incident will travel faster at that position than one passing just after the incident. If  $v(d, t, l)$  is the vehicular speed in link  $l$  at time  $t$  on date  $d$ , Zhu et al. proposed the following four

**Table 1: Notation**

Notation	Definition
$K$	Number of traffic states.
$k$	Index of a traffic state.
$S$	Number of segments.
$s$	Index of a segment.
$x_{sn}$	$n$ -th data in the $s$ -th segment.
$N_s$	Number of observations in the $s$ -th segment.
$\theta_k$	Parameter of the $k$ -th distribution.
$\pi_s$	Mixing coefficient vector in segment $s$ .
$\Lambda$	$(\{\pi_s\}_{s=1, \dots, S}, \{\theta_k\}_{k=1, \dots, K})$ .
$\sigma(s, x)$	Traffic state in $s$ when $x$ was observed.
$d(s)$	Usual traffic state in $s$ .
$d(s, x)$	Divergence of $\sigma(s, x)$ from $d(s)$ .
$X_s$	Set of data observed in the $s$ -th segment, i.e., $X_s = \{x_{s1}, x_{s2}, \dots, x_{sN_s}\}$ .
$X$	Whole set of data, i.e., $X = \{X_1, \dots, X_S\}$ .
$X_c$	Data sequence from car $c$ , i.e., $X_c = \langle (s_1, x_1), (s_2, x_2), \dots, (s_{N_c}, x_{N_c}) \rangle$ .
$D(X_c)$	Divergence of $X_c$ .

features:  $v(d, t, l)$ ,  $v(d, t, l) - v(d, t - 1, l)$ ,  $v(d, t, l - 1)$  and  $v(d, t, l + 1) - v(d, t, l)$ , where link  $l - 1$  is the next link upstream of  $l$ , and  $l + 1$  is the next link downstream. These feature vectors are filtered using the heuristics above and analyzed by distance-based outlier detection. In another AID study, Akatsuka et al. [2] proposed an alternative feature vector. From the viewpoint of machine learning, AID can be regarded as a classification problem. Abdulhai et al. [1] used neural networks, and Yuan et al. [18] used support vector machines, to classify the observed vectors from stationary sensors as being incident based or otherwise. AID can also be regarded as an application of the change-point detection problem in time-series analysis, with Wang et al. [13] developing a hybrid method using time-series analysis and machine learning.

In this paper, we regard the AID problem as an anomaly detection problem. Previous work exploits characteristics of congested traffic, such as slowdown, in which vehicular speed decreases even in the absence of a traffic incident. We take another approach to follow the intuitive meaning of “anomaly”; namely, an event different than usual. For this purpose, the traffic should be described by a probabilistic model. We therefore exploit the idea of probabilistic topic models, which was originally studied in the field of natural language processing [5, 4]. The proposed method estimates both a set of traffic states over an entire route and the mixing coefficients for each road segment, with a traffic state corresponding to a topic.

## 3. METHODOLOGY

Table 1 summarizes the notations used in this paper.

This section describes our traffic state model and incident detection method. We first introduce a method for applying a probabilistic topic model to PCD. Our task is to estimate the model parameters using a PCD archive and to identify incidents by comparing the usual and current traffic states, which are obtained from the learned model.

### 3.1 Traffic State Model

Intuitively, we can identify some traffic states as “smooth”

or “congested” regardless of location. Vehicles travel fast in smooth states and behave in a stop-and-go fashion in heavily congested states. When observing the speed of a probe car, the value is likely to be small if the car is in “congested traffic,” or large if the traffic is “smooth.” The value will also be affected by geographical conditions, such as curves and slopes. In short, the behavior of a car is affected by the surrounding traffic state, and the observed values for the probe car will change, whereas the traffic state is latent and varies according to the time and place. This relation between traffic states and PCD can be modeled using the latent Dirichlet allocation [5], the simplest topic model [4].

Traffic states are strongly related to roads, so we introduce the *segment* as the unit for watching traffic. The segment is defined independently of the PCD by the spatiotemporal space of observation. For example, one such segment could be defined as the section between Interchanges A and B on the inbound direction of Route 3 between 6 a.m. and 9 a.m. PCD includes timestamps and location data, that are obtained via GPS and are represented by longitude and latitude, and each probe-car observation can be assigned to a predefined segment.

PCD also has information on values such as speed and direction that can be recorded directly in the PCD or calculated using sequential observation. Here, all the observations are aggregated for each segment, and a set  $X_s$  of the observed data for the  $s$ -th segment is obtained. The symbol  $x_{sn}$ , the  $n$ -th value of  $X_s$ , might have either a scalar or a vector value. For simplicity in this paper, we assumed that  $x_{sn}$  was a scalar value, but our method could be extended to observe vector values.

Our model associates a traffic state with a probability distribution. Let  $K$  be the number of states, with the  $k$ -th traffic state corresponding to the parameter  $\theta_k$ . The probability distribution for the  $s$ -th segment, given by  $p(x|s)$ , is described in terms of a mixture of these  $K$  distributions and can be described as follows:

$$p(x|s) = \sum_{k=1}^K \pi_{sk} p(x|\theta_k), \quad (1)$$

where  $\pi_{sk}$  is the mixing coefficient for the  $k$ -th state and satisfies the conditions:

$$0 \leq \pi_{sk} \leq 1, \quad \sum_{k=1}^K \pi_{sk} = 1 \quad (2)$$

for each  $s$ . The state parameters  $\{\theta_1, \dots, \theta_K\}$  are identical for all segments, but the mixing coefficient vector  $\boldsymbol{\pi}_s = (\pi_{s1} \cdots \pi_{sK})^\top$  is different for each segment. By using a global  $\theta_k$ , we can compare and characterize segments in terms of local  $\boldsymbol{\pi}_s$ . For example, straight sections are dominated by “smooth” states, with sections that include tollgates that are dominated by “congested” states.

Finally, for each segment, the generative process for this model was as follows.

1. Choose a hidden state  $k \sim$  multinomial probability distribution  $\text{Multi}(\boldsymbol{\pi}_s)$ .
2. Generate the value  $x_{sn} \sim p(x_{sn}|\theta_k)$ .

### 3.2 Parameter Estimation

Our model is described by a mixture distribution, with its maximum-likelihood parameters estimated by an EM al-

gorithm, using  $X$  as training data [3]. For simplicity, we introduce the symbol  $\Lambda$  as a set of all parameters in the model. For the entire set  $X$  of observed data, the likelihood under the model introduced above is given by the following equation:

$$L(X) = \prod_{s=1}^S \prod_{n=1}^{N_s} \sum_{k=1}^K \pi_{sk} p(x_{sn}|\theta_k). \quad (3)$$

The update equations are derived by considering the maximization of the following  $Q$  function under constraint (2):

$$Q(X, \Lambda, \hat{\Lambda}) = \sum_{s=1}^S \sum_{n=1}^{N_s} \sum_{k=1}^K p(k|x_{sn}, \hat{\Lambda}) \log p(k, x_{sn}|\Lambda), \quad (4)$$

where

$$p(k|x_{sn}, \hat{\Lambda}) = \frac{\hat{\pi}_{sk} p(x_{sn}|\hat{\theta}_k)}{\sum_{k=1}^K \hat{\pi}_{sk} p(x_{sn}|\hat{\theta}_k)} \equiv \gamma_{snk} \quad (5)$$

$$p(k, x_{sn}|\Lambda) = \pi_{sk} p(x_{sn}|\theta_k), \quad (6)$$

and  $\hat{\Lambda}$  refers to the parameters estimated in the previous EM iteration.

This  $Q$  is maximized by introducing Lagrange multipliers and setting its partial derivative to zero. The update equation for  $\theta_k$  is then derived by solving the equation:

$$\sum_{s=1}^S \sum_{n=1}^{N_s} \frac{\gamma_{snk}}{p(x_{sn}|\theta_k)} \frac{\partial}{\partial \theta_k} p(x_{sn}|\theta_k) = 0. \quad (7)$$

For example, assume a Poisson distribution for  $p$  when any  $x_{sn}$  values, e.g., speed, are nonnegative integers, then:

$$p(x_{sn}|\theta_k) \equiv p(x_{sn}|\lambda_k) = \frac{\lambda_k^{x_{sn}} e^{-\lambda_k}}{x_{sn}!}, \quad (8)$$

where  $\lambda_k$  is both the mean and variance, and is the only parameter of  $p$ . In this case, by solving equation (7), the update equation for  $\lambda_k$  is derived as:

$$\lambda_k = \frac{\sum_{s=1}^S \sum_{n=1}^{N_s} \gamma_{snk} x_{sn}}{\sum_{s=1}^S \sum_{n=1}^{N_s} \gamma_{snk}}. \quad (9)$$

For the mixing coefficient  $\boldsymbol{\pi}_s$  for the  $s$ -th segment, we obtain, regardless of  $p$ , the equation:

$$\pi_{sk} = \frac{\sum_{n=1}^{N_s} \gamma_{snk}}{N_s}. \quad (10)$$

We now have the EM algorithm for estimating the parameters of our traffic state model: After generating  $\Lambda$  at random, the EM iteration alternates between the E step, which calculates all  $\gamma_{snk}$  using equation (5), and the M step, which updates  $\Lambda$  according to equations (7) and (10), until the log likelihood  $\log L(X)$  converges.

### 3.3 Incident Detection

We have now described our traffic state model and its parameter estimation method. Given the estimated parameter



	G:Good, M:Moderate, S:Stop								
$\sigma(s)$	G	G	G	G	G	M	S	G	G
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
$\sigma(s, x)$	G	G	M	S	S	S	S	G	
$X_c$	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$x_6$	$x_7$		route
segment	1	2	3	4	5	6	7	8	9

Figure 1: Concept of divergence comparison

$\Lambda$  and the value  $x$  observed in segment  $s$ , the posterior distribution is given by  $p(k|x, s)$ . We now define the *current traffic state* when  $x$  was observed, denoted by  $\sigma(s, x)$ , as the maximum probable state given  $x$ . Using the posterior distribution with Bayes' theorem,  $\sigma(s, x)$  is estimated as:

$$\sigma(s, x) = \arg \max_k \{\pi_{sk} p(x|\theta_k)\}. \quad (11)$$

Meanwhile, the learned model itself reflects the usual state over the whole observation period because the parameters are estimated to fit the distribution in the dataset. We can therefore define the *usual traffic state* for the  $s$ -th segment, denoted by  $\sigma(s)$ , as the maximum probable state:

$$\sigma(s) = \arg \max_k \pi_{sk}. \quad (12)$$

We now have the usual and the current traffic states for each segment. Figure 1 describes our idea of incident detection via *divergence comparison*. For example, the usual state  $\sigma(s)$  may indicate smooth traffic in a straight mid-night segment, congested traffic in a rush-hour segment, or stop-and-go traffic in segments that contain tollgates for any time of day. If  $\sigma(s)$  indicates congested traffic and  $\sigma(s, x)$  is also congested, the current traffic remains usual and would not be considered an anomaly. If the usual state  $\sigma(s)$  indicates free-flowing traffic and the current state  $\sigma(s, x)$  indicates stop-and-go traffic, then it would be suspected that an anomaly caused by an incident has occurred.

Our AID method measures the degree of anomaly for each probe car's trajectory. Assume that a probe car  $c$  traverses a road, observing a set of  $N_c$  values. Let  $X_c$  be the sequence of data such that each is a tuple of segment and value observed by  $c$  as described in Table 1. Let  $x_n$  be an observed value in a segment  $s_n$ . Of course, we can count how many times  $\sigma(s_n, x_n)$  differs from  $\sigma(s_n)$ , but this approach regards major differences in the same light as minor differences, which perhaps stem from individual variation rather than from a traffic incident. We therefore introduce the *divergence* of the current state from the usual state, denoted by  $d(s_n, x_n)$ , to quantify the difference between the two states. Because in our model each state is associated with a probability distribution, we measure this difference in terms of the Kullback–Leibler divergence of the current state's distribution from the usual state's distribution. The  $k$ -th state corresponds to the probability distribution  $p(x|\theta_k)$ , and therefore:

$$d(s_n, x_n) = \sum_x p(x|\theta_{\sigma(s_n, x_n)}) \log \frac{p(x|\theta_{\sigma(s_n, x_n)})}{p(x|\theta_{\sigma(s_n)})}, \quad (13)$$

where  $p$  is discrete. For example, assume Poisson distribu-



Figure 2: Three routes of Shuto Expressway within the Tokyo area

tion for  $p$  as equation (8). The divergence is derived as:

$$d(s_n, x_n) = \lambda_{\sigma(s_n)} - \lambda_{\sigma(s_n, x_n)} + \lambda_{\sigma(s_n, x_n)} \log \frac{\lambda_{\sigma(s_n, x_n)}}{\lambda_{\sigma(s_n)}}. \quad (14)$$

The behavior of a car  $c$  is determined as anomalous if the estimated state behind the observed data sequence  $X_c$  is quite different from the usual state. We define the divergence of  $X_c$  from the usual state, denoted by  $D_{\text{all}}(X_c)$ , as:

$$D_{\text{all}}(X_c) = \sum_{n=1}^{N_c} d(s_n, x_n). \quad (15)$$

The more a car behaves differently from its usual behavior, the larger  $D_{\text{all}}(X_c)$  will be.  $D_{\text{all}}(X_c)$  is considered as a score of the degree of anomaly, with  $c$  being determined as anomalous when  $D_{\text{all}}(X_c)$  is sufficiently large, i.e., larger than a predefined threshold.

There are two points to consider about  $D_{\text{all}}(X_c)$ . First,  $D_{\text{all}}(X_c)$  will also increase the longer the car  $c$  runs, and any car would eventually be determined as being anomalous. We therefore define the normalized divergence  $D(X_c)$  as the sum of the largest  $N$  divergences  $d(s_n, x_n)$  if  $N_c$  is not less than  $N$ . Otherwise,  $D(X_c)$  is equivalent to  $D_{\text{all}}(X_c)$ . We have used  $D$  instead of  $D_{\text{all}}$  in the rest of the paper. Second, when a car generates values periodically, no observation or multiple observations in a trajectory can be assigned to a single segment. Our idea of divergence comparison in Figure 1 assumed that one segment corresponded to one current state, which might require interpolation or aggregation of data for each segment.

## 4. EXPERIMENT

### 4.1 Dataset and Preprocessing

Our probe-car dataset was obtained from probe cars traveling on three routes on the Shuto Expressway system in Tokyo during 2011. The route information is displayed in Figure 2, with the three routes being shown as thick red lines on a map of Tokyo.

Data preprocessing comprised four phases: 1) segment definition, 2) map matching, 3) trajectory identification, and 4) interpolation. These procedures are described below.

### 4.1.1 Segment Definition

Traffic state information is strongly related to geographical conditions. We defined road segments by partitioning each route on the expressway every 50 m for estimation at a finer level of granularity. The direction was noted. This experiment did not consider temporal partitioning, even though the traffic in some places changed considerably over time. Therefore, each segment represented a certain 50-m length of roadway for a certain direction on a certain expressway route, and all data for a segment were treated without any consideration of time.

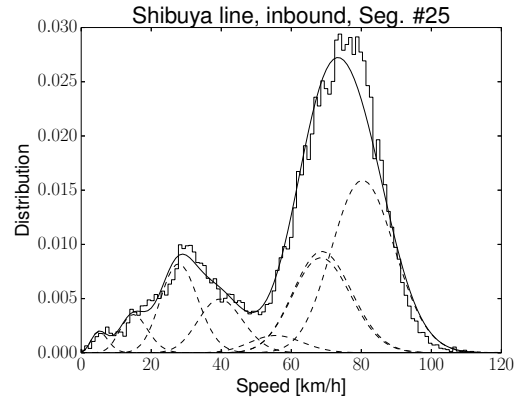
### 4.1.2 Map Matching

Despite the above definition of a segment being based on an expressway route, location data in PCD were described in terms of the two-dimensional (2-D) coordinates of longitude and latitude, with the original observation not being related to any particular segment. It was therefore necessary to identify the segment that the probe car was in from the time and position for every observation, even though in this experiment we did not consider the timestamps. Map matching is a technology for identifying the road segment on which the vehicle is traveling and for locating the vehicle within that segment [9], and several methods have been proposed [14, 16]. In this experiment, map matching was conducted in the simplest way: a probe car’s observation was matched with the nearest segment to the car’s location. The direction was estimated from the angular difference between the probe car’s heading azimuth in the PCD and the segment’s azimuth for each direction, and then choosing the direction that gave the smaller angle.

### 4.1.3 Trajectory Identification

After map matching, each probe-car observation whose location was represented by coordinates in the 2-D space was matched with the nearest-neighbor segment, as defined in the first phase of preprocessing. However, the observations form a collection of punctuated data, with each observation being separate from the others. Therefore, the continuous movement of the car, i.e., its trajectory, is not directly available. To identify trajectories, we grouped all observations in the probe-car dataset by the car’s ID and sorted them by timestamp for each group, before concatenating them in chronological order whenever the time gap between two consecutive observations was 10 min or less. A probe car does not always travel the entire length of a route, because it can enter or exit the route at intermediate junctions. For a car traveling on a single route, its trajectory can be visualized in terms of a time–space diagram [12]. Figure 5 is an example of such a diagram and will be described in detail later.

After the trajectory identification, we labeled each trajectory, using the traffic log made available by the administrator of the Shuto Expressway. This traffic log is recorded via stationary sensors on or alongside the roads every 5 min, together with notations about incidents such as accidents and construction. A trajectory was labeled as anomalous whenever a car passed a stationary sensor that had recorded an incident at that time. Table 2 summarizes the statistical information for our probe-car dataset after trajectory identification. The number of anomalies means the number of trajectories for a probe car passing the scene of an incident when the incident occurred but does not indicate the number of unique incidents.



**Figure 3: Histogram of speed of probe cars in a segment and estimated Poisson mixture**

### 4.1.4 Interpolation

We used a probe car’s speed as the observed value in this experiment. However, as mentioned in Section 3.3, our detection method estimated the current state for each trajectory for each segment that the car had passed. Our 50-m segment was too short for fast-moving probe cars to conduct observations in every segment, whereas a slow-moving car generated multiple data in a single segment. We therefore formed an observation sequence for a trajectory by linear interpolation, giving a sequence of consecutive observations at 50-m intervals.

## 4.2 Parameter Estimation

In this experiment, the observed values represented the speed of probe cars as nonnegative integers. We therefore assumed a Poisson distribution for the probability distribution corresponding to each traffic state.

We also assumed  $K$ , the number of traffic states, to be 8. In a preliminary experiment, we estimated the parameters of our traffic model while varying the value of  $K$  up to 100, and we used the Akaike information criterion (AIC) to evaluate the model. However, the effect of  $K$  was substantially less than that of the likelihood for improving the AIC, with AIC being almost the same regardless of  $K$ . If  $K$  is assumed to be large, there is a tendency for multiple states to have almost the same distribution.

We implemented the EM algorithm described in Section 3.2 using OpenMP for multiprocessing. The estimation was executed on our 32-core Xeon computer for each route of the Shuto Expressway. It took about 1 min for each direction of the Shibuya and Shinjuku routes, and about 2 min for each direction of the Ikebukuro route. Figure 3 shows the actual histogram for a segment of the inbound Shibuya route as a step line chart and the estimated Poisson mixture as a solid curved line. Each of the eight Poisson distributions was multiplied by the mixing coefficients  $\pi_{sk}$ , and each is also shown in Figure 3 as dashed curves. The estimated curve almost fits the actual histogram for the training dataset.

## 4.3 Incident Detection

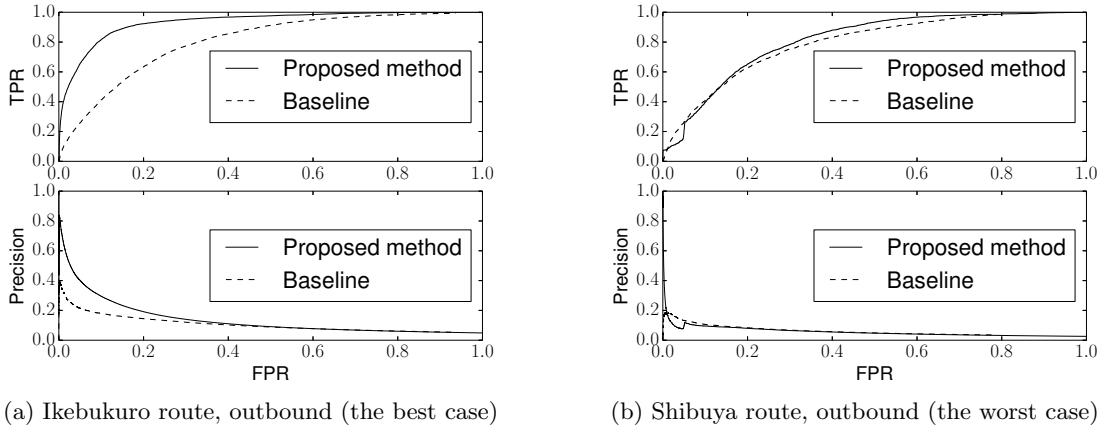
Using the estimated traffic model, we examined whether the proposed method could identify anomalous trajectories. We calculated the divergence for each trajectory and sorted the trajectories in order of their divergence. The divergence

**Table 2: Statistics on trajectories in our probe-car dataset**

Routes	Shibuya route		Shinjuku route		Ikebukuro route	
	Inbound	Outbound	Inbound	Outbound	Inbound	Outbound
Period	January 1, 2011 – December 31, 2011 (365 days)					
# of trajectories	100,581	95,386	95,293	88,345	128,789	114,942
# of anomalies	4,259	2,475	4,365	3,891	6,089	5,603
Average travel distance [km]	5.7	5.8	6.4	6.7	7.5	8.1

**Table 3: AID results**

Routes		Shibuya route		Shinjuku route		Ikebukuro route	
		Inbound	Outbound	Inbound	Outbound	Inbound	Outbound
AUC	Our method	0.912	0.812	0.927	0.919	0.902	0.933
	Baseline [20]	0.802	0.794	0.846	0.780	0.823	0.805

**Figure 4: ROC curves (upper frames) and precision vs. false positive rate (lower frames)**

of a trajectory was calculated by summing the top  $N$  divergences among the observations. In a preliminary experiment, we conducted the detection for several values of  $N$ , obtaining the best result when  $N$  was 20. Because we were using 50-m segments, the divergences of trajectories were normalized to 1-km equivalents.

For comparison, we implemented a second method based on Zhu et al. [20], which was described in Section 2. The method was modified to enable its application to our dataset, and although it detected outlier segments represented by the pair of time and position, our system was evaluated in terms of anomalous cars. Therefore, we judged that a detection event was successful if the detected car was labeled as an anomaly in our dataset, even if the detected segment for the detected car was not a segment involving an incident.

Our detection method gives an alert when the divergence of a trajectory exceeds a given threshold, and the compared method gives an alert when the average distance of a feature vector from other vectors exceeds a given threshold. The lower the threshold, the more alerts will be issued. We evaluated the selectivity performance of the two methods in terms of a receiver-operating characteristic (ROC) curve. An ROC curve is drawn by plotting the true positive rate (TPR), which is equivalent to recall, against the false positive rate (FPR). The area under the curve (AUC) indicates the discrimination performance, with larger AUC values indicating better discrimination.

The results are displayed in Table 3 and Figure 4. Table 3 reports the AUC of the proposed and baseline methods on

our probe-car datasets. The results showed that our method had better selectivity for cars that have passed incident locations, despite using fewer heuristics about anomalies than the baseline method. Figure 4 shows the ROC curves in the upper frames and the precision against FPR in the lower frames. Although the ROC curve should connect points (0,0) and (1,1), that of the baseline method broke off before (1,1) was reached, because the method filtered out some feature vectors, with the number of subject trajectories being less than the total number of trajectories. The AUC of the baseline method was calculated by interpolating linearly between the right-hand end of the ROC curve and (1,1). Figure 4(a) shows the curves for the outbound Ikebukuro route, which was the best case in our experiment. The precision exceeded 80% for the worst 1,000 trajectories. Figure 4(b) shows the curves for the outbound Shibuya route, which was the worst case.

Figure 5 shows examples of trajectories for the outbound Shibuya route that had much divergence in terms of their time-space diagram. Each plot shows the position of a probe car against time. The position is represented as the distance from the origin of the line: the bottom corresponds to the Tokyo interchange, the westernmost along the Shibuya route, and the top corresponds to the Tanimachi (easternmost) junction. Horizontal pink lines indicate the positions of interchanges and junctions. The inbound direction is the direction from the bottom to the top in this diagram. Therefore, trajectories downward and to the right involve traveling along the outbound Shibuya route. The color of the plot

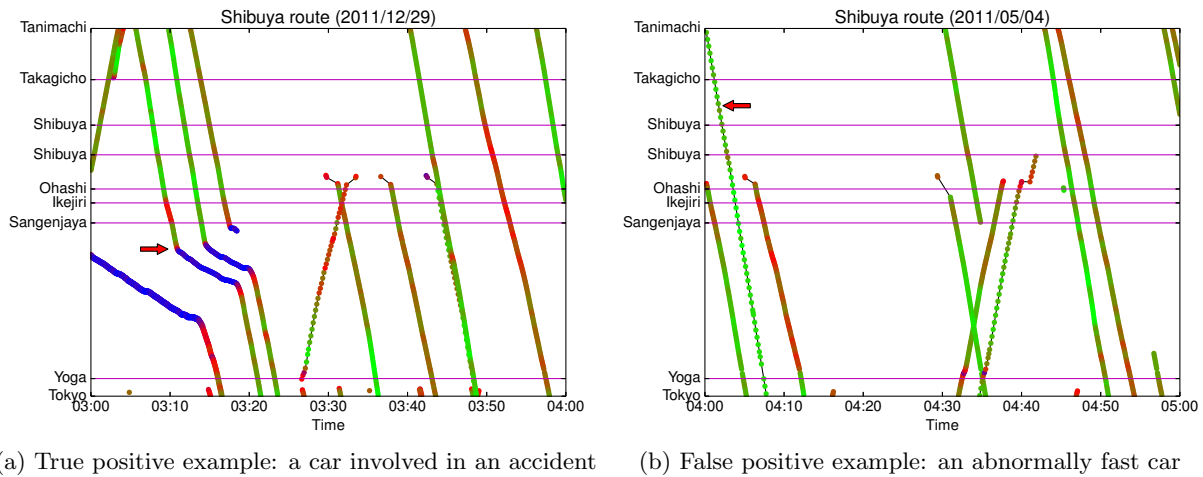


Figure 5: Time–space diagrams for probe cars

indicates the speed of the probe car at that point. Green represents high speed (100 km/h), red is moderate speed (50 km/h), and blue is “almost stopped” (0 km/h). The color changes gradually according to the speed. The trajectory marked with an arrow in Figure 5(a) was the most anomalous trajectory, with this car being directly affected by an incident. The diagram shows that this car was “stop-and-go” between Sengenjaya and Yoga. On the other hand, the marked trajectory in Figure 5(b) was ranked as no. 301 among the anomalous trajectories. This car did not encounter an incident. The diagram shows the car traveling rapidly along the route.

## 5. DISCUSSION

In Figure 4(b), the TPR of the proposed method was sluggish when the TPR was around 0.1, indicating that the proposed method rarely detected anomalous cars correctly even when the threshold was lowered to some degree. Cars whose trajectories became anomalous at this point traveled rapidly, with the car of the marked trajectory in Figure 5(b) being an example of such cars. This car traversed the route before dawn, when the traffic is usually smooth. One of the possible reasons for such false positives is that our experiment did not consider temporal partitioning in the segment definition, even though the traffic changed considerably over time. The spatial length of a segment, as well as parameters  $K$  and  $N$ , should also be determined in future studies.

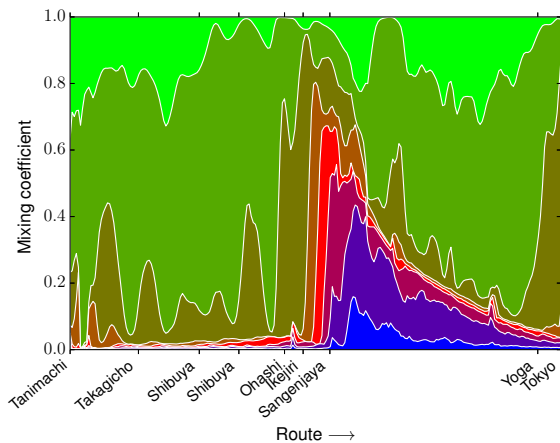
The following discussion demonstrates another analysis on the abovementioned false positives based on the estimated traffic states. In this experiment, we used the speed of the probe car, and the traffic state was represented by the Poisson distribution, which was characterized by the mean and variance parameter  $\lambda$ . The stacked area chart in Figure 6 shows the estimated mixing coefficients for the eight Poisson distributions for each segment of the outbound Shibuya route. The horizontal axis shows the position along the route, and cars travel from left to right. The colored areas show the mixing coefficient for each state varying with position. They are in order of  $\lambda$ , with the bottommost being the slowest, and the topmost being the fastest. From Tanimachi to Ikejiri, the top three fastest states were dominant, which means that cars usually travel quickly in this

section. However, from Ikejiri to Yoga, the coefficients for the faster states decrease as the slower states begin to dominate, because the cars usually travel more slowly in this section. Therefore, although the marked trajectory in Figure 5(b) does not seem to include any incidents, this behavior was quite different from the usual running pattern, and our method identified this as an anomaly. It is noteworthy that our traffic state model has enabled this sort of analysis, with every segment being characterized using a single set of traffic states. Although we used the data sequence to give observations at 50-m intervals for each probe car, stationary sensors can also generate similar data except for tracking information for each car. Because parameter estimation does not require such information, this road characteristics analysis can be conducted using stationary sensors, and its output might be applied to other problems; e.g., route guidance.

The Shuto Expressway system has many bottlenecks, such as curves and narrow sections that involve frequent changes in vehicular speed, unlike freeways. We speculate that this is the reason that our intuitive method found that “unusual” car behavior worked better than a heuristic method that pays attention to changes in speed. On the other hand, a significantly fast car can be surely determined as an anomaly if its behavior is statistically unusual relative to the past observations, although this kind of “unusualness” is not a problem for drivers. Anomalies accompanying a slowdown in vehicular speed can be regarded as a subset of the anomalies discussed in this paper. The administrator and drivers have the option to filter the outcome of our detection algorithm using additional heuristics. However, a particular incident is hard to detect by the proposed method if the traffic behavior in the incident is just like regular spontaneous congestion. We are currently conducting investigations into detailed issues as a further study, expanding our dataset sphere from only three routes to all the routes on the Shuto Expressway system.

## 6. CONCLUSION

We have studied the problem of detecting traffic incidents using probe-car data. Although congestion can be detected by monitoring vehicular speeds, it is chronic in some spots and does not necessarily indicate the occurrence of an in-



**Figure 6: Mixing coefficients for eight Poisson distributions for each segment of the outbound Shibuya route**

cident. To detect traffic incidents, we propose an approach that compares the current traffic state with the usual one for that location in terms of anomalous car movements, using a probabilistic topic model to describe the state of monitored traffic. We proposed an incident detection method that measured the difference between the usual and current states. Our method was applied to real probe-car data that were collected on the Shuto Expressway system in Tokyo, and the discrimination performance was evaluated. The results showed that our method could discriminate trajectories affected by incidents from other trajectories, although abnormally fast cars were also reported as anomalies, giving a low precision for certain routes.

## 7. ACKNOWLEDGMENTS

The traffic log used in our experiment as the ground truth for incident occurrence was made available by Metropolitan Expressway Co., Ltd.

## 8. REFERENCES

- [1] B. Abdulhai and S. G. Ritchie. Enhancing the universality and transferability of freeway incident detection using a bayesian-based neural network. *Transportation Research Part C: Emerging Technologies*, 7(5):261–280, 1999.
- [2] H. Akatsuka, A. Takasu, K. Aihara, and J. Adachi. Highway incident detection based on probe car data. In *International Conference on Information Systems (Information Systems 2013)*, pages 103–110, 2013.
- [3] C. M. Bishop. *Pattern Recognition and Machine Learning*, chapter 9. Springer, 2006.
- [4] D. M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, Apr. 2012.
- [5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.
- [6] C. de Fabritiis, R. Ragona, and G. Valenti. Traffic estimation and prediction based on real time floating car data. In *Proceedings of the 11th International IEEE Conference on Intelligent Transportation Systems*, pages 197–203, 2008.
- [7] East Nippon Expressway Co., Ltd. Generation mechanism of traffic congestion caused by traffic concentration. <http://www.e-nexco.co.jp/activity/safety/mechanism.html>. Accessed: 2013-12-06.
- [8] B. Kerner, C. Demir, R. Herrtwich, S. L. Klenov, H. Rehborn, M. Aleksic, and A. Haug. Traffic state detection with floating car data in road networks. In *Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems*, pages 44–49, 2005.
- [9] M. A. Quddus, W. Y. Ochieng, and R. B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15(5):312–328, 2007.
- [10] J. M. Sussman. ITS: A short history and a perspective on the future. In *Perspectives on Intelligent Transportation Systems (ITS)*, pages 3–17. Springer US, 2005.
- [11] S. Tao, V. Manolopoulos, S. Rodriguez Duenas, and A. Rusu. Real-time urban traffic state estimation with a-gps mobile phones as probes. *Journal of Transportation Technologies*, 2(1):22–31, 2012.
- [12] M. Treiber and A. Kesting. *Traffic Flow Dynamics*. Springer Berlin Heidelberg, 2013.
- [13] J. Wang, X. Li, S. Liao, and Z. Hua. A hybrid approach for automatic incident detection. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1176–1185, 2013.
- [14] C. E. White, D. Bernstein, and A. L. Kornhauser. Some map matching algorithms for personal navigation assistants. *Transportation Research Part C: Emerging Technologies*, 8(1):91–108, 2000.
- [15] J. Xia, W. Huang, and J. Guo. A clustering approach to online freeway traffic state identification using its data. *KSCE Journal of Civil Engineering*, 16(3):426–432, 2012.
- [16] J.-s. Yang, S. Kang, and K.-s. Chon. The map matching algorithm of gps data with relatively long polling time intervals. *Journal of the Eastern Asia Society for Transportation Studies*, 6:2561–2573, 2005.
- [17] J. Yoon, B. Noble, and M. Liu. Surface street traffic estimation. In *Proceedings of the 5th international conference on Mobile systems, applications and services, MobiSys ’07*, pages 220–232, New York, NY, USA, 2007. ACM.
- [18] F. Yuan and R. L. Cheu. Incident detection using support vector machines. *Transportation Research Part C: Emerging Technologies*, 11(3–4):309–328, 2003. Traffic Detection and Estimation.
- [19] Y. Yuan, J. W. C. Van Lint, R. Wilson, F. van Wageningen-Kessels, and S. Hoogendoorn. Real-time lagrangian traffic state estimator for freeways. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):59–70, 2012.
- [20] T. Zhu, J. Wang, and W. Lv. Outlier mining based automatic incident detection on urban arterial road. In *Proceedings of the 6th International Conference on Mobile Technology, Application & Systems, Mobility ’09*, pages 29:1–29:6, New York, NY, USA, 2009. ACM.

# Predictive Trip Planning – Smart Routing in Smart Cities

Thomas Liebig  
TU Dortmund University  
Artificial Intelligence Group  
Dortmund, Germany  
thomas.liebig@tu-dortmund.de

Christian Bockermann  
TU Dortmund University  
Artificial Intelligence Group  
Dortmund, Germany  
christian.bockermann@tu-dortmund.de

Nico Piatkowski  
TU Dortmund University  
Artificial Intelligence Group  
Dortmund, Germany  
nico.piatkowski@tu-dortmund.de

Katharina Morik  
TU Dortmund University  
Artificial Intelligence Group  
Dortmund, Germany  
katharina.morik@tu-dortmund.de

## ABSTRACT

Smart route planning gathers increasing interest as cities become crowded and jammed. We present a system for individual trip planning that incorporates future traffic hazards in routing. Future traffic conditions are computed by a Spatio-Temporal Random Field based on a stream of sensor readings. In addition, our approach estimates traffic flow in areas with low sensor coverage using a Gaussian Process Regression. The conditioning of spatial regression on intermediate predictions of a discrete probabilistic graphical model allows to incorporate historical data, streamed online data and a rich dependency structure at the same time. We demonstrate the system and test model assumptions with a real-world use-case from Dublin city, Ireland.

## Categories and Subject Descriptors

G.3 [Probability and Statistics]: Multivariate statistics, Stochastic processes, Time series analysis; H.4.2 [Information Systems Applications]: Types of Systems—*Logistics*; J.7 [Computer in Other Systems]: Real time

## 1. INTRODUCTION

The incentive for the creation of smart cities is the increase of living quality and performance of the city. This is often accompanied with various mobile phone apps or web services to bring new services to the people of a city – advertising events, spreading city information or guiding people to their destinations by providing smart trip planning based on the city’s spirit.

With the unpleasant trend of growing congestion in modern urban areas, smart route planning becomes an essential

service in the smart city development. Existing trip planning systems consider current traffic hazards and historical speed profiles which are recorded by personal position traces and mobile phone network data [27]. The traffic message channel (TMC) is a radio service that transmits hazards to personal navigation devices. Due to technical limitation it can just address locations which are situated foremost at inter-urban highways [15]. Besides the limited spatial granularity of TMC and its broadcast of past traffic states, TMC is a phasing out technology as the advent of digital radio supersedes submission of RDS-TMC messages via VHF/FM [32].

The fast moving traffic situations in urban areas demand for a thorough routing that incorporates as fresh information about the city’s infrastructure as possible. This work presents an approach to *situation dependent trip planning* that incorporates real time information gained from smart city sensors and combines this data with a model for estimating future traffic situations for route calculation. The proposed system provides three components: (1) an interactive web-based user interfaces that is based on the popular *OpenTripPlanner* project [22]. The web interface allows for users to specify start and target location and triggers the route planning and provides a REST-ful service (REpresentation State Transfer, introduced in [26]) interface to integrate such services into mobile applications. (2) A real-time backend engine, based on the *streams* framework [6], which provides data stream processing for various types of data. We provide input adapters for *streams* to read and process SCATS data [1] emitted from automatic traffic loops (city sensors). This allows us to maintain an up-to-date view of the city’s current traffic state. (3) A sophisticated dynamic traffic model that is integrated into the backend stream engine and which provides traffic flow estimation at unobserved locations at future times.

The combination of these components is a trip planner that incorporates the latest traffic state information as well as using a fine-grained future traffic flow estimation for urban trip planning. We test our trip planner in a use case scenario in the city of Dublin. The city is amongst the most jammed cities in Europe [2]. The city holds about 630 SCATS sensors, each providing current traffic flow and

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.



vehicle speed at the sensor location.

The paper is structured as follows. In the second section we describe the general architecture of the presented system regarding the input and output of the trip planner, the data analysis and the stream processing connecting middleware. The third section deals with the application of our proposed trip planner to a use case in Dublin, Ireland. In the fourth section we provide a discussion of the work together with future directions. The fifth section presents related work.

## 2. GENERAL ARCHITECTURE

We give an overview of the system developed to address the veracity, velocity and sparsity problems of urban traffic management. The system has been developed as part of the INSIGHT project. This section describes the input and output of the system, the individual components that perform the data analysis, and the stream processing connecting middleware.

### 2.1 System Components

As already noted in the introduction, we built the system aiming real time streaming capabilities. Based on the *streams* framework, the core engine is a data flow graph that models the data stream processing of the incoming SCATS data. This graph can easily be defined by means of the *streams* XML configuration language and features the integration of custom components directly into the data flow graph. As can be seen in Figure 1, this data flow graph contains the SCATS data source as well as several nodes that represent preprocessing operations. A crucial component within that stream processing is our Spatio-Temporal Random Field (STRF) implementation<sup>1</sup>, which is used in combination with the sensor readings to provide a model for traffic flow prediction.

With the *service layer* API provided by *streams*, we export access to the traffic prediction model to the OpenTripPlanner component. The OpenTripPlanner provides the interface to let the user specify queries for route planning. Based on a given query  $(v, w)$  with a starting location  $v$  and a destination  $w$ , it computes the optimal route  $v \rightarrow p_0 \dots p_k \rightarrow w$  based on traffic costs. Here we plug in a cost-model for the routing that is based on the traffic flow estimation and the current city infrastructure status. This cost-model is queried by OpenTripPlanner using the *service layer* API.

### 2.2 Traffic Model

The key component of our system is the traffic model. It combines two machine learning methods in a novel way, in order to achieve traffic flow predictions for nearly arbitrary locations and points in time. This traffic model addresses multiple facets of the trip planning problem:

- sparsity of stationary sensor readings among the city,
- velocity of real-time traffic readings and computation, and
- veracity of future traffic flow predictions.

Based on a stream of observed sensor measurements, a Spatio-Temporal Random Field [25] estimates the future sensor values, whereas values for non-sensor locations are estimated

<sup>1</sup>The C++ implementation of STRF and the JNI interface can be found at: <http://sfb876.tu-dortmund.de/strf>

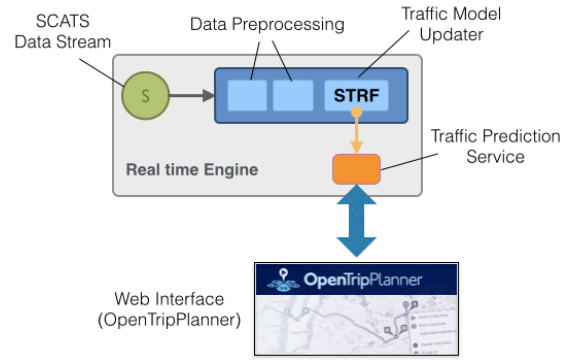


Figure 1: A general overview of the components of the predictive trip planning system. The real time engine continuously manages a up-to-date state of the city infrastructure and exports the traffic estimator as prediction service to the OpenTripPlanner. Best viewed in color.

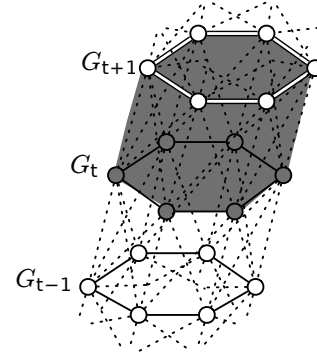


Figure 2: Simple spatio-temporal graph. The underlying spatio graph  $G_0$  is a simple circle of 6 nodes.

using Gaussian Processes [20]. To the best of the authors knowledge, streamed STRF+GP prediction has not been considered until now and is therefore a novel method for traffic modelling. A comparable method is proposed in the same workshop [29] that combines a linear dynamic system with Gaussian Processes for near-time forecasts. Comparing these two models in terms of precision and speed is open for future work.

### Spatio-Temporal Random Field for Flow Prediction

In order to model the temporal dynamics of the traffic flow as measured by the SCATS sensors (Figure 5), a Spatio-Temporal Random Field is constructed. The intuition behind STRF is based on sequential probabilistic graphical models, also known as linear chains, which are popular in the natural language processing community. There, consecutive words or corresponding word features are connected to a sequence of labels that reflects an underlying domain of interest like entities or part of speech tags. If a sensor network, represented by a spatial graph  $G_0 = (V_0, E_0)$ , is considered that generates measurements over space and



time, it is appealing to identify the joint measurement of all sensors with a single word in a sentence and connect those structures to form a temporal chain  $G_1 - G_2 - \dots - G_T$ . Each part  $G_t = (V_t, E_t)$  of the temporal chain replicates the given *spatial graph*  $G_0$ , which represents the underlying physical placement of sensors, i.e., the spatial structure of random variables that does not change over time. The parts are connected by a set of spatio-temporal edges  $E_{t-1,t} \subset V_{t-1} \times V_t$  for  $t = 2, \dots, T$  and  $E_{0,1} = \emptyset$ , that represent dependencies between adjacent snapshot graphs  $G_{t-1}$  and  $G_t$ , assuming a Markov property among snapshots, so that  $E_{t,t+h} = \emptyset$  whenever  $h > 1$  for any  $t$ . The resulting spatio-temporal graph  $G$ , consists of the snapshot graphs  $G_t$  stacked in order for time frames  $t = 1, 2, \dots, T$  and the temporal edges connecting them:  $G := (V, E)$  for  $V := \cup_{t=1}^T V_t$  and  $E := \cup_{t=1}^T \{E_t \cup E_{t-1,t}\}$ . This construction is shown in Figure 2. There, a simple circle of 6 nodes serves as spatial graph  $G_0$ .

Finally,  $G$  is used to induce a generative probabilistic graphical model that allows us to predict (an approximation to) each sensors maximum-a-posterior (MAP) state as well as the corresponding marginal probabilities. The full joint probability mass function is given by

$$p_{\theta}(\mathbf{X} = \mathbf{x}) = \frac{1}{\Psi(\theta)} \prod_{v \in V} \psi_v(\mathbf{x}) \prod_{(v,w) \in E} \psi_{(v,w)}(\mathbf{x}).$$

Here,  $\mathbf{X}$  represents the random state of all sensors at all  $T$  points in time and  $\mathbf{x}$  is a particular assignment to  $\mathbf{X}$ . It is assumed that each sensor emits a discrete value from a finite set  $\mathcal{X}$ . By construction, a single vertex  $v$  corresponds to a single SCATS sensor  $s$  at a fixed point in time  $t$ . The potential function of an STRF has a special form that obeys the smooth temporal dynamics inherent in spatio-temporal data.

$$\psi_v(\mathbf{x}) = \psi_{s(t)}(\mathbf{x}) = \exp \left\langle \sum_{i=1}^t \frac{1}{t-i+1} \mathbf{Z}_{s,i}, \phi_{s(t)}(\mathbf{x}) \right\rangle$$

The STRF is therefore parametrized by the vectors  $\mathbf{Z}_{s,i}$  that store one weight for each of the  $|\mathcal{X}|$  possible values for each sensor  $s$  and point in time  $1 \leq i \leq T$ . The function  $\phi_{s(t)}$  generates an indicator vector that contains exactly one 1 at the position of the state that is assigned to sensor  $s$  at time  $t$  in  $\mathbf{x}$  and zero otherwise. For a given data set, the parameters  $\mathbf{Z}$  are fitted by regularized maximum-likelihood estimation.

As soon as the parameters are learned from the data, predictions can be computed via MAP estimation,

$$\hat{\mathbf{x}} = \arg \max_{\mathbf{x}_{V \setminus U} \in \mathcal{X}} p_{\theta}(\mathbf{x}_{V \setminus U} \mid \mathbf{x}_U), \quad (1)$$

where  $U \subset V$  is a set of spatio-temporal vertices with known values. The nodes in  $U$  are termed observed nodes. Notice that  $U = \emptyset$  is a perfectly valid choice that yields the most probable state for each node, given no observed nodes. To compute this quantity, the sum-product algorithm [17] is applied, often referred to as loopy belief propagation (LBP). Although LBP computes only approximate marginals and therefore MAP estimation by LBP may not be perfect [14], it suffices our purpose.

## Gaussian Process Model for Flow Imputation

We model the junction based traffic flow values within a Gaussian Process regression framework, similar to the approach in [20]. In the traffic graph each junction corresponds to one vertex. To each vertex  $v_i$  in the graph, we introduce a latent variable  $f_i$  which represents the true traffic flow at  $v_i$ . The observed traffic flow values are conditioned on the latent function values with Gaussian noise  $\epsilon_i$

$$y_i = f_i + \epsilon_i, \epsilon_i \sim \mathcal{N}(0, \sigma^2). \quad (2)$$

We assume that the random vector of all latent function values follows a Gaussian Process (GP), and in turn, any finite set of function values  $\mathbf{f} = f_i : i = 1, \dots, M$  has a multivariate Gaussian distribution with mean and covariances computed with mean and covariance functions of the GP. The multivariate Gaussian prior distribution of the function values  $\mathbf{f}$  is written as

$$P(\mathbf{f} \mid \mathbf{X}) = \mathcal{N}(0, K), \quad (3)$$

where  $K$  is the so-called kernel and denotes the  $M \times M$  covariance matrix, zero mean is assumed without loss of generality.

For traffic flow values at unmeasured locations  $u$ , the predictive distribution can be computed as follows. Based on the property of GP, the vector of observed traffic flows ( $v$  at locations  $-u$ ) and unobserved traffic flows ( $f_u$ ) follows a Gaussian distribution

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_u \end{bmatrix} \sim \mathcal{N} \left( 0, \begin{bmatrix} \hat{K}_{-u,-u} + \sigma^2 I & \hat{K}_{-u,u} \\ \hat{K}_{u,-u} & \hat{K}_{u,u} \end{bmatrix} \right), \quad (4)$$

where  $\hat{K}_{u,-u}$  are the corresponding entries of  $\hat{K}$  between the unobserved vertices  $u$  and observed ones  $-u$ .  $\hat{K}_{-u,-u}$ ,  $\hat{K}_{u,u}$ , and  $\hat{K}_{-u,u}$  are defined equivalently.  $I$  is an identity matrix of size  $| -u |$ .

Finally the conditional distribution of the unobserved traffic flows are still Gaussian with the mean  $m$  and the covariance matrix  $\Sigma$ :

$$m = \hat{K}_{u,-u} (\hat{K}_{-u,-u} + \sigma^2 I)^{-1} \mathbf{y} \\ \Sigma = \hat{K}_{u,u} - \hat{K}_{u,-u} (\hat{K}_{-u,-u} + \sigma^2 I)^{-1} \hat{K}_{-u,u}.$$

Since the latent variables  $\mathbf{f}$  are linked together in a graph  $\mathcal{G}$ , it is obvious that the covariances are closely related to the network structure: the variables are highly correlated if they are adjacent in  $\mathcal{G}$ , and vice versa. Therefore we can employ graph kernels [31] to denote the covariance functions  $k(x_i, x_j)$  among the locations  $x_i$  and  $x_j$ , and thus the covariance matrix.

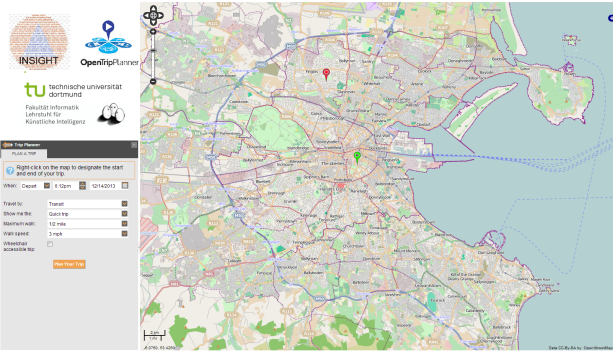
The work in [20, 19] describes methods to incorporate knowledge on preferred routes in the kernel matrix. Lacking this information, we decide for the commonly used regularized Laplacian kernel function

$$K = [\beta(L + I/\alpha^2)]^{-1}, \quad (5)$$

where  $\alpha$  and  $\beta$  are hyperparameters.  $L$  denotes the combinatorial Laplacian, which is computed as  $L = D - A$ , where  $A$  denotes the adjacency matrix of the graph  $\mathcal{G}$ .  $D$  is a diagonal matrix with entries  $d_{i,i} = \sum_j A_{i,j}$

## 2.3 OpenTripPlanner

OpenTripPlanner (OTP) is an open source initiative for route calculation. The traffic network for route calculation



**Figure 3: OpenTripPlanner User Interface.** Map view is on the right side including a green pin which indicates the start location and a red pin that indicates the target. Best viewed in color.

is generated using data from OpenStreetMap and (eventually) public transport schedules. Thus, OpenTripPlanner allows route calculation for multiple modes of transportation including walking, bicycling, transit or its combinations. However, vehicular routing is possible, but for data quality reasons in OpenStreetMap concerning the turning restrictions [28] it is not advisable.

The default routing algorithm in OTP is the A\* algorithm [13] which utilizes a cost-heuristic to prune the Dijkstra search [8]. At every considered intermediate location (between start and target location) the cost-heuristic estimates a lower bound of the remaining travel costs to the target. The cost estimate for traversing this intermediate location is calculated using the sum of the costs to the location and the estimated remaining costs.

OpenTripPlanner consists of two components an API and a web application which interfaces the API using RESTful services. The API loads the traffic network graph, and calculates the routes. The web application provides an interactive browser based user interface with a map view. A user of the trip planner can form a trip request by selecting a start and a target location on the map, see Figure 3 for a Screenshot of the user interface. Besides the web application there exist OpenTripPlanner user interfaces for mobile devices. The variety of existing user interfaces stresses the sustainability of our decision for OpenTripPlanner.

## 2.4 The streams Framework

The need for real time capabilities in today’s data processing and the steady decrease of latency from data acquisition to knowledge extraction or information use from that data led to a growing demand for general purpose stream processing environments. Several such frameworks have evolved – *Storm*, *Kafka* or Yahoo!’s *S4* engine are among the most popular open-source approaches to streaming data. They all feature slightly different APIs and come with slightly different philosophies. Focusing on a more middle-layer approach is the *streams* framework proposed in [6], which aims at providing a light-weight high-level abstraction for defining data flow networks in an easy-to-use XML configuration. It comes with its own execution engine, but also features the transparent execution of data flow graphs on existing engines such

as *Storm*. We base our decision for the *streams* framework on its recent applications that highlight its high throughput capabilities [9] and the built-in data mining operators [5].

### SCATS Data Processing with streams

Within the *streams* framework, a data source is represented as a sequences of data items, which in turn are sets of key-value pairs, i.e. event attributes and their values. Processes within a *streams* data flow graph consume data items from streams and apply functions onto the data. The data flow graph for manipulation, analysis and filtering of the streams is formulated in an XML-based language that *streams* provides. A sample XML configuration is given in Figure 4.

```
<container>
  <stream id="scats:data" url="http://..."
    class="eu.insight.input.ScatsStream" />

  <process input="scats:data">
    <!-- .. custom functions .. -->
    <eu.insight.data.DataNormalization />
    <eu.insight.traffic.TrafficEstimator
      id="predictor" />
  </process>
</container>
```

**Figure 4: XML representation of a streams container with a source for SCATS data and a process that applies a normalization to each data item and then forwards it to a traffic estimation processor.**

The process setup of Figure 4 defines a single data source that provides a stream of SCATS sensor data. A *process* is attached to this source and continuously reads items from that source. For each of the data item, it applies a sequence of custom functions (so called *processors*) that reflect data transformations or other actions on the items. In the example above, we include a SCATS specific *DataNormalization* step as well as our custom *TrafficEstimator* implementation directly into the data flow graph.

### Service Level API

The *streams* runtime provides a simple RMI-based service invocation of data flow components that do provide remote services. The *TrafficEstimator* defines such a remote interface and is automatically registered as a service with identifier “*predictor*”. This allows service methods of that estimator to be asynchronously called from outside the data flow graph, i.e. from within our modified *OpenTripPlanner* component.

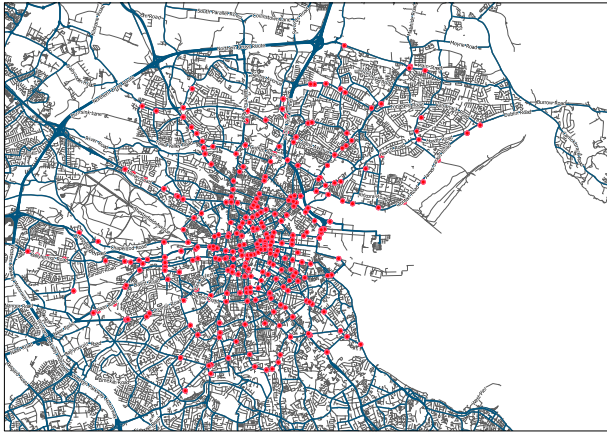
The service method that is defined by the *TrafficEstimator* is exactly the cost-retrieval function that is required within the A\* algorithm of the *OpenTripPlanner*:

$$getCost(x, y, t)$$

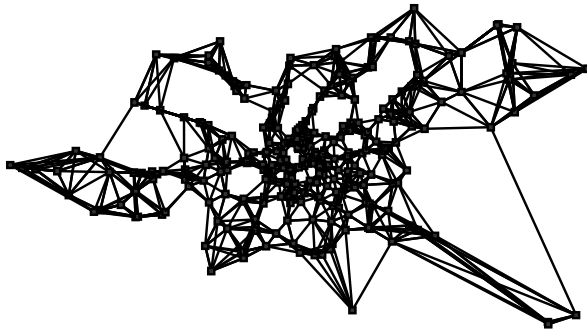
where  $x$  and  $y$  are the longitude and latitude of the location and  $t$  is the time at which the traffic flow for  $(x, y)$  shall be predicted.

## 3. EMPIRICAL EVALUATION

In this section we present the application of our proposed trip planner to a use case in Dublin, Ireland. We used real data streams obtained from the SCATS sensors of Dublin



**Figure 5: Locations of SCATS sensors (marked by red dots) within Dublin, Ireland. Best viewed in color.**



**Figure 6: Spatial graph  $G_0$  that is derived from the SCATS sensor locations. Each vertex is connected to its 7 nearest neighbors in order to include short- and long-distance dependencies.**

city. The stream was collected between January and April 2013 and comprises  $\approx 9$ GB of data. The SCATS dataset includes 966 sensors, see Figure 5 for their spatial distribution among the traffic network. SCATS sensors transmit information on traffic flow every six minutes. The data set is publicly available<sup>2</sup>.

For the experiments in Dublin, the traffic network is generated based on the OpenStreetMap<sup>3</sup> data. In the preprocessing step the network is restricted to a bounding window of the city size. Next, every street is split at any junction in order to retrieve street segments. In result we obtain a graph that represents the traffic network. The SCATS locations, are mapped to their nearest neighbours within this street network.

In the preprocessing step the sensor readings are aggregated within fixed time intervals. We tested various intervals and decided for 30 minutes, as lower aggregates are too noisy, caused by traffic lights and sensor fidelity.

The spatial graph  $G_0$  that is required for the STRF is con-

structed as  $k$ -nearest-neighbor ( $k$ NN) graph of the SCATS sensor locations. In what follows, a 7NN graph (Figure 6) is used, since a smaller  $k$  induces graphs with large disconnected components and a larger  $k$  results in more complex models without improving the performance of the method. The fact that no information about the actual street network is used to build  $G_0$  might seem counterintuitive, but undirected graphical models like STRF do not use or rely on any notion of flow. They rather make use of conditional independence, i.e. the state of any node  $v$  can be computed if the states of its neighboring nodes are known. Thus, the  $k$ NN graph can capture long-distance dependencies that are not represented in the actual street network connectivity. The maximum traffic flow value that is measured by each SCATS sensor in each 30-minutes-window is discretized into one of 6 consecutive intervals. A separate STRF model for each day of the week is constructed and each day is further partitioned into 48 snapshot graphs, since we can divide a day into 48 blocks of 30 minutes length. The model parameters are estimated on SCATS data between January 1 and March 31 2013 and evaluated on data from April 2013.

The evaluation data is streamed as observed nodes into the STRF which computes a new conditioned MAP prediction (Equation 1) for all unobserved vertices of the spatio-temporal graph  $G$  whenever time proceeds to the next temporal snapshot. The discrete predictions are then de-discretized by taking the mean of the bounds of the corresponding intervals and subsequently forwarded to the Gaussian Process which uses these predictions to predict values at non-sensor locations. Notice that although the discretization with subsequent de-discretization seems inconvenient at a first glance, it allows the STRF to model any non-linear temporal dynamics of the sensor measurements, i.e. the flow at a fixed sensor might change instantly if the sensor is located close to a factory at shift changeover.

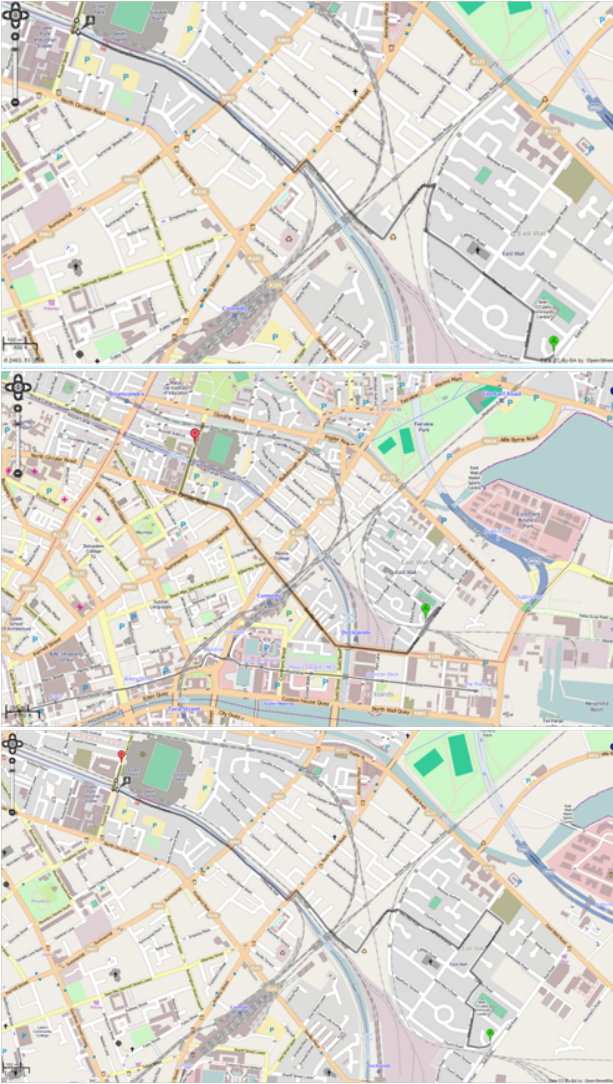
Application of Gaussian Processes requires a joint multivariate Gaussian distribution among the considered random variables. In our case, these random variables denote the traffic flow per junction. Literature on traffic flow theory [18, 7] tested traffic flow distributions and supports a hypothesis for a joint lognormal distribution. We test our dataset for this hypothesis. Thus, we apply the Mardia [21] normality test to the preprocessed data set. The test checks multivariate skewness and kurtosis. We apply the implementation contained in the R package MVN [16]. The tests confirmed the hypothesis that the recorded traffic flow (obtained from the SCATS system) is lognormal distributed. Thus, application of Gaussian Processes to log-transformed traffic flow values is possible. The hyper-parameters for the GP are chosen in advance using a grid search. Best performance was achieved with  $\alpha = 1/2$  and  $\beta = 1/2$ . The STRF provides complete knowledge on future sensor readings which is necessary for our GP. As the STRF model performs well [25], we set the noise among the sensor data in our GP to a small variance of 0.0001. For easy tractability, we set the GP up to model about 5000 locations among the city of Dublin.

The OpenTripPlanner creates a query for the costs at a particular coordinate in space-time. The query is transmitted from the route calculation to the traffic model. There, the query is matched to the discrete space. The spatial coordinates are encoded in the WGS84 reference system [24]. To avoid precision problems during the matching between the components, the spatial coordinate is matched with a

<sup>2</sup>Dublin SCATS data: <http://www.dublinked.ie>

<sup>3</sup>OpenStreetMap: <http://www.openstreetmap.org>





**Figure 7: Results of route calculations for fixed start and target at different timestamps (from top to bottom: 7:00, 8:00, 8:30). Best viewed in color.**

nearest neighbour method using a KDTree data structure [23]. The nearest neighbor matching offers also the possibility to query costs for arbitrary locations. The timestamp of the query is discretized to one of the 48 bins we applied in the STRF.

We apply our trip planner for a particular Monday in data set (8th April 2013) and compute routes from a fixed start to a fixed target at different time stamps. Figure 7 shows that different routes are calculated depending on the traffic situation.

#### 4. DISCUSSION AND FUTURE WORK

Within this paper we presented a novel approach for trip planning in highly congested urban areas. Our approach computes intelligent routes that avoid traffic hazards which did not yet occur. The proposed trip planner consists of a

continuous traffic model based on real-time sensor readings and a web based user interface. We combined the real-time traffic model and the trip calculation with a streaming backbone. We applied the trip planner to a real-world use case in the city of Dublin, Ireland. The city is amongst the most congested ones and jam avoidance is a natural goal of the citizens.

Our traffic model combines latest advances in traffic flow estimation. On the one hand, prediction of future sensor values is performed with a spatio-temporal random field, which is trained in advance. Based on these estimates, the traffic flow for unobserved locations is performed by a Gaussian Process Regression. We successfully applied the Regularized Laplacian Kernel. In literature, also other kernels have been successfully applied to the problem, [19, 30]. Exploration of different kernel methods is subject for future research.

The route calculation component of our approach is based upon the OpenTripPlanner project as it provides a separation among the trip planner and the user interface. The OpenTripPlanner interface for mobile devices<sup>4</sup> guides the direction for further extension of our approach to a personal navigation device.

We perform trip calculation with the A\* algorithm, a speedup using contraction hierarchies (a speedup heuristic that introduces shortcuts in the traffic network, compare [11]) is promising. This allows the extension to multi-modal trip planning (compare [4]) and computation in embedded devices. Prediction of delays in the public transport network are another important direction for multi-modality.

Besides the SCATS data also other data sources provide useful information for dynamic cost estimation. The integration of bus travel times or user generated (crowdsourcing and social network) data in our model is possible by dynamically changing the traffic network (in case of road blockages) or introducing dynamic weights (in case of an accident or flooding on a street segment). Future studies need to explore these directions.

One still might argue that if all people use our trip planner and all people use the same alternative way to avoid a jam it will occur somewhere else. This hypothesis needs to be validated. The effect might not be so strong as the individual persons do not start at the same time and do not have same start and target locations thus traffic distributes differently among the traffic network. If our STRF model is updated regularly the jams might be prevented. Another path, we follow in future is individual route calculation, which adds some minor perturbations to the route in order to avoid occurrence of unexpected jams that result from route delivery.

The real-world application of the trip planner was performed as part of the INSIGHT project [3]. Aim of the European funded project (grant number 318225) is not just congestion reduction, but also the real-time prediction of upcoming hazards and proactive control. The city of Dublin is subject to many floods that cause problems for urban traffic. Our trip planner is basis for further extensions that avoid flooded areas based on flood observations and predictions.

#### 5. RELATED WORK

Previous sections already discussed related approaches. Here, we present briefly recent work on dynamic cost es-

<sup>4</sup>OpenTripPlanner for Android: <https://github.com/cutr-at-usf/opentripplanner-for-android/wiki>

timization for trip planning in smart cities. Recent work [10] addresses travel time forecasts based on the delays in the public transportation system. Main drawback of their method is that buses have extra lanes at most junctions and their movement follows a regular pattern. The inclusion of traffic loop readings was motivated in their section on future work. The dynamic traffic flow estimation is a major problem in traffic theory. Common approach is the usage of a  $k$ -Nearest Neighbour algorithm which calculates traffic flow estimates as weighted average of the  $k$  nearest observations [12]. In contrast, our approach models future traffic flow values based on their temporal patterns, correlations and dependencies. Foremost, our model requires less memory as  $k$ -NN which has to store all previously seen sensor values for continuous traffic flow estimation. Another paper that compares two prediction models for traffic flow estimation is presented in [29]. By combining a Gauss Markov Model with a Gaussian Process, their work provides a faster model which is suitable for near time predictions (as required for automatic signal control). The model estimates future values by consecutive application of the model. In contrast, the hereby presented work estimates all future time slices at once. In result, we could build the valuable trip planner application on top of the traffic estimation model and highlighted its usability. Improvement of the estimation method, and comparison of estimation accuracy is subject for future work.

## 6. ACKNOWLEDGMENTS

This research has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement number 318225, INSIGHT – “Intelligent Synthesis and Real-time Response using Massive Streaming of Heterogeneous Data”. Additionally, this work has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource-Constrained Data Analysis”, project A1. We acknowledge Dublin city council and Dominik Dahlem for data collection and preparation of the SCATS dataset. We thank Jakub Marecek for assistance with the OpenTripPlanner project, and the anonymous reviewers for their inspiring feedback.

## 7. REFERENCES

- [1] SCATS. *Sydney Coordinated Adaptive Traffic System*, Available: <http://www.scats.com.au/> [Last accessed: 27 June 2013], 2013.
- [2] TomTom European Congestion Index. *TomTom*, Available: <http://www.tomtom.com/lib/doc/congestionindex/2013-0322-TomTom-CongestionIndex-2012-Annual-EUR-mi.pdf> [Last accessed: 26 June 2013], 2013.
- [3] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Gunopulos, and D. Kinane. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *Proceedings of the 17th International Conference on Extending Database Technology*, page (to appear), 2014.
- [4] H. Bast, M. Brodesser, and S. Storandt. Result Diversity for Multi-Modal Route Planning. In D. Frigioni and S. Stiller, editors, *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 33 of *OpenAccess Series in Informatics (OASISs)*, pages 123–136, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [5] C. Bockermann and H. Blom. Processing Data Streams with the RapidMiner Streams-Plugin. In *Proceedings of the 3rd RapidMiner Community Meeting and Conference*, 2012.
- [6] C. Bockermann and H. Blom. The streams framework. Technical Report 5, TU Dortmund University, 12 2012.
- [7] G. Davis. estimation theory approach to monitoring and updating average daily traffic. Technical Report mn/rc 97-05, minnesota department of transportation, office of research administration, january 1997.
- [8] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.
- [9] A. Gal, S. Keren, M. Sondak, M. Weidlich, H. Blom, and C. Bockermann. Grand challenge: The techniball system. In *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, DEBS '13, pages 319–324, New York, NY, USA, 2013. ACM.
- [10] L. Gasparini, E. Bouillet, F. Calabrese, O. Verscheure, B. O'Brien, and M. O'Donnell. System and analytics for continuously assessing transport systems from sparse and noisy observations: Case study in dublin. In *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*, pages 1827–1832, 2011.
- [11] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In C. McGeoch, editor, *Experimental Algorithms*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin Heidelberg, 2008.
- [12] X. Gong and F. Wang. Three Improvements on KNN-NPR for Traffic Flow Forecasting. In *Proceedings of the 5th International Conference on Intelligent Transportation Systems*, pages 736–740. IEEE Press, 2002.
- [13] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.
- [14] U. Heinemann and A. Globerson. What cannot be learned with bethe approximations. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, Barcelona, Spain, 2011.
- [15] ISO 14819-1:2003. Traffic and Traveller Information (TTI) – TTI messages via traffic message coding – Part 1: Coding protocol for Radio Data System – Traffic Message Channel (RDS-TMC) using ALERT-C. International Organization for Standardization, 2003.
- [16] S. Kormaz. *MVN: Multivariate Normality Tests*, 2013. R package version 1.0.
- [17] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE*

- Transactions on Information Theory*, 47(2):498–519, 2001.
- [18] G. Lay. *Handbook of Road Technology, Fourth Edition*. taylor & francis, 2009.
- [19] T. Liebig, Z. Xu, and M. May. Incorporating mobility patterns in pedestrian quantity estimation and sensor placement. In J. Nin and D. Villatoro, editors, *Citizen in Sensor Networks*, volume 7685 of *Lecture Notes in Computer Science*, pages 67–80. Springer Berlin Heidelberg, 2013.
- [20] T. Liebig, Z. Xu, M. May, and S. Wrobel. Pedestrian quantity estimation with trajectory patterns. In P. A. Flach, T. Bie, and N. Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 7524 of *Lecture Notes in Computer Science*, pages 629–643. Springer Berlin Heidelberg, 2012.
- [21] K. V. Mardia. Measures of multivariate skewness and kurtosis with applications. *Biometrika*, 57:519–530, 1970.
- [22] B. McHugh. The opentripplanner project. Technical Report Metro RTO Grant Final Report, TriMet, August 2011.
- [23] A. Moore. An introductory tutorial on kd-trees. Technical Report Technical Report No. 209, Computer Laboratory, University of Cambridge, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1991.
- [24] National Imagery and Mapping Agency. Department of Defense World Geodetic System 1984: its definition and relationships with local geodetic systems. Technical Report TR8350.2, National Imagery and Mapping Agency, St. Louis, MO, USA, january 2000.
- [25] N. Piatkowski, S. Lee, and K. Morik. Spatio-temporal random fields: compressible representation and distributed estimation. *Machine Learning*, 93(1):115–139, 2013.
- [26] L. Richardson and S. Ruby. *RESTful Web Services*. O’Reilly Series. O’Reilly Media, Incorporated, 2007.
- [27] R.-P. Schäfer. IQ Routes and HD Traffic: Technology Insights About Tomtom’s Time-dynamic Navigation Concept. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE ’09, pages 171–172, New York, NY, USA, 2009. ACM.
- [28] S. Scheider and J. Possin. Affordance-based individuation of junctions in open street map. *Journal of Spatial Information Science*, 4(1):31–56, 2012.
- [29] F. Schnitzler, T. Liebig, S. Mannor, and K. Morik. Combining a gauss-markov model and gaussian process for traffic prediction in dublin city center. In *Proceedings of the Workshop on Mining Urban Data at the International Conference on Extending Database Technology*, page (to appear), 2014.
- [30] B. Selby and K. M. Kockelman. Spatial prediction of traffic levels in unmeasured locations: applications of universal kriging and geographically weighted regression. *Journal of Transport Geography*, 29:24–32, May 2013.
- [31] A. Smola and R. Kondor. Kernels and regularization on graphs. In *Proc. Conf. on Learning Theory and Kernel Machines*, pages 144–158, 2003.
- [32] TISA Executive Office. Provision of a free minimum universal traffic information service. Technical Report EO12004, The Traveller Information Services Association, May 2012.

# Addressing the Sparsity of Location Information on Twitter

Dimitrios Kotzias  
University of Athens  
dkotzias@di.uoa.gr

Ted Lappas  
Stevens Institute of  
Technology  
tlappas@stevens.edu

Dimitrios Gunopoulos  
University of Athens  
dg@di.uoa.gr

## ABSTRACT

Micro-blogging services such as Twitter have gained enormous popularity over the last few years leading to massive volumes of user generated content. In combination with the proliferation of smart-phones, this information is generated *live* from a *multitude* of content contributors. Interestingly, the content and timestamp of tweets is not the only information that can produce useful knowledge. The location information of users is of great significance since it can be utilized in a variety of applications such as emergency identification, tracking the spread of a disease and advertising. Unfortunately, information regarding location is very rare since many users do not accurately specify their location, and fewer posts have geographic coordinates. In this work, we aim to confront this data sparsity issue. Utilizing Twitter's social graph and content, we are able to obtain users from a specific location. We optimize our method to work with minimum amount of queries considering the large volume of data in such settings. We also provide a mechanism for geo-locating a tweet within a city and present the qualitative enrichment in our data, achieved by our method.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Data Mining; J.4 [Social and Behavioural Sciences]: Sociology

## General Terms

Algorithms

## Keywords

Social Networks, Data Sparsity, Location Profiling

## 1. INTRODUCTION

Over the last decade on-line platforms where individuals generate and contribute content have gained massive popularity. In most of these platforms individuals are connected, establishing social networks. Such networks attracted interest from various scientific fields due to the numerous re-

search challenges and the plethora of potential applications. A typical example is Twitter currently hosting more than 200 million users contributing more than 400 million tweets per day [12]. Twitter is unique among the social networks since it propagates information to an immense amount of people very quickly, in fact many times, faster than conventional news networks [23]. Combined with the increasing popularity of smart phones, and mobile internet availability, users generate content from a variety of locations in real time, creating a very diverse and spatio-temporally spread-out network of information.

Naturally, with the formation of such a network, there has been an increase in research as well as applications for spatio-temporal data [18, 13, 8]. These applications utilize the content, time of creation as well as location of data. Location information is important as it can transfer knowledge from the online back to the real world, and aid towards personalized and localized information services. It is invaluable for a variety of applications such as discovering the way diseases spread [6], emergency identification and response [25], localized event detection and relevant news propagation [28], analysis of the behavioural patterns and mobility of people within a city [8, 10] as well as online advertising [1]. Furthermore, social networks themselves are also adopting a local-focus philosophy, with examples of Twitter recently starting reporting local trends<sup>1</sup>, while search engines consider your location among other factors when returning results<sup>2</sup>.

Despite the fact that user contributed content in such platforms is always characterized by a well defined timestamp, unfortunately location information is very sparse. Research in Twitter suggests that a big percentage of users either do not provide their location information in their profiles, or input noisy data [7], with only 48% of users providing an actual location with city or lower level accuracy [11]. Moreover, the number of tweets with geographical coordinates is much lower, in the order of 1% [26]. Reasonably, this sparsity of information constitutes a major issue for all the aforementioned applications that require it.

In this paper, we address the problem of sparsity of location data by providing a framework that utilizes information from the content of a users' tweets, as well as the social graph around her. Considering the large volume of data in such settings and the fact that our system works with the Twitter API, we take into account the limitations set by such systems and optimize our method for the minimum amount of queries. Our contribution is threefold:

<sup>1</sup><https://blog.twitter.com/2010/now-trending-local-trends>

<sup>2</sup><http://www.google.com/landing/now/>



- We formalise the problem of location identification using both social graph information and text information as a resource utilization optimization problem in the context of the limitation in the Twitter system.
- We analyse the locality of the social graph in Twitter, and the connectivity between users who live in geographical proximity.
- We provide a framework for (i) identifying more users at a location of a city-level granularity, (ii) attaching geographical coordinates to individual tweets within that city.

In addition to the precision of our model, we also perform quantitative and qualitative analysis on how the increase in users, and thus tweets, enriches our data by extracting topic models and analysing them. We chose to apply our method on Twitter due to (i) its increased academic interest, (ii) large volume of active users and tweets and (iii) its unique nature as a fast information and events propagator.

## 2. RELATED WORK

Related work can be broadly divided into research done for identifying the location of a user, or the location of each individual tweet.

In the first category, Eisenstein et al. attempt to solve the problem through geographical topic models [9]. They capture the difference in the use of language for a specific topic, between people from distant areas. They are able to predict the location of a user with an error mean distance of 900 km, and achieve a 27% accuracy when predicting the state of a user. Their results indicate that people have different ways of discussing a topic in different areas, however, there has to be a significant distance between these areas. Within a city level granularity, people are more likely to have similar conversational habits. More recently, Ahmed et al. in [2] proposed a tree-like hierarchical structure of topics, at which the lower levels of the tree, represent more specific versions of the general topics at the parent nodes. This way, they are able to extract location specific topics, and place users with an average error of 298 kilometers in the same dataset. Cheng et al. in [7] utilized the locality of phrases rather than topics, and manage to pinpoint the city of 51% of users, within 100 miles of their actual location. Mahmud et. al. in [22] improve upon this method by identifying *named local words*. These terms are very local terms such as the name of a location or the name of places, retrieved from services such as foursquare. They use a Multinomial Naive Bayes classifier and test hierarchical algorithms that first predict the country and then the state of a user, to estimate the city of a user. They achieve an accuracy of 58% for 100 miles radius.

These approaches take advantage of the difference in the use of language about specific topics or words which identify the location of users, however in cases of smaller areas, where there are not many language differences, these methods would not perform well. Our work differentiates at two fundamental levels. Firstly, we take advantage of a users writings as well as the relationships in the social graph, which enables us to predict location with greater geographical detail. Secondly we also propose a way to attach exact geographical coordinates in a tweet level, after we have identified the location of the user who created it.

In the second category, Ikawa et. al in [14] attempt to estimate the location of a tweet by associating expressions with locations. For each query tweet, they find the location with the closest word list and place it there. However, the underlying assumption that people will tweet about the place they are in, and then express their feelings about it, does not always hold, and they achieve an accuracy of 14% for a radius of 5 kilometres. Li et, al achieve a better precision for the same problem as they attempt to identify Places of Interest (POI) a tweet may belong to. [21]. They build a Language Model for each POI, based on tweets that occurred there and information crawled from websites, and then rank the KL-divergences for each query tweet, to identify the candidate POI's. They test their method for the 10 top POIs at a city, and reach an accuracy of more than 60% for their best case. However, their accuracy fluctuates greatly based on the number of tweets about a POI, and their premise is somehow unrealistic, since in a real-world scenario, there are much more than 10 possible locations within a city. Kinsella et. al provide a framework, which is closer to our work since it is used to pinpoint the location of both users and tweets in a variety of granularities in [15]. They build language models for each location and test a Query Likelihood model, in order to predict the location of a tweet. Their best results accomplish an accuracy of 31.9% for users for a town granularity and 13.9% for tweet location in zip-code granularity. In their work, they use the same model to solve both problems, while we discriminate between the two and mostly take advantage of the social graph for user location prediction and term-models for tweet location prediction, which enables us to create a more robust system.

Ren et. al [24] take the social graph into account in order to identify a users' location. They place each user to the location of the majority of his friends. They achieve a precision of 59.3%, however they only test it for 704 users. Working on a much larger dataset from facebook data, Backstrom et. al [4] suggest a correlation between friendship and distance on the map, and build a more elaborate model to find the probability of a users home location, given the location of his contacts. They place 67.7% of users correctly, however, both these methods, assume that we know all the friends of the user we are trying to locate, which in very large graphs such as those created by social media is rarely the case. Our method aims to minimize the number of queries at the social graph, and more importantly operates without the knowledge of the location of all the friends of a user.

To the best of our knowledge, despite the large number of relevant papers which focus on estimating the location of a single user, the problem of identifying more users from a specific location, has yet to be solved. In contrast to most of the previous work done in this area we: (i) attempt to identify the location of a user in a *city level* accuracy, which is much more limited than 100 miles radius, set this far, (ii) combine this information to attach geographical coordinates to a tweet, by taking under consideration *both* the relationships of users in the social graph as well as the content of their writings, and (iii) use a computational model based on the Twitter system, optimized for the number of queries.

## 3. PROBLEM DEFINITION

Location data in twitter are rare, and given the value of such information we attempt to discover more users at a specific location. We analyse data about a city as whole, and

we set two interconnected goals: (i) Discover more users that live in that city and (ii) create a system that is able to attach exact coordinates on a Tweet level within that city. Moreover, we assume that we incur a *cost* each time we make a query about the connections or the location of a user. This assumption is pragmatic because social networks have limitations on how many times one can inquire about such information within a specific time frame. For instance Twitter only allows for 15 queries per 15 minutes for the friends of user. Taking under consideration the size and increase rate of social networks today, this constitutes a practical constrain as well, since it is computationally expensive to process the entire graph. The problems are formally defined below:

**Problem 1 - User Level**

Given set of users  $U$  who tweeted from a specific city  $C$ , identify as many users from the same city as possible, by asking at most  $k$ -queries about the social graph.

**Problem 2 - Tweet Level**

Given a set of tweets  $T_1$  from a specific city  $C$  for which we know their exact geographical location, and a set  $T_2$  for which we do not have such information, attach geographical coordinates to  $T_2$ .

We consider the two problems to be interconnected, because by following a hierarchical scheme, the tweets of users identified in  $C$ , can be considered as  $T_2$  and be geo-located on the map, with the solution to problem 2.

**4. PRELIMINARIES**

**4.1 Twitter**

Twitter consists of messages which have a maximum length of 140 characters, which may or may not include location information. With the proliferation of mobiles phones, many users *tweet* from a variety of locations. However, there is a wide array of topics and uses for a tweet, which in combination with the limited text size, render most of them impossible to analyse and categorize. Users have the ability to have a static location in their profiles, however, according to [11] the location field of Twitter users many times is empty or contains inaccurate or in-comprehensive information.

**4.2 Graph Analysis and Motivation**

Previous work has indicated that there is a correlation between proximity on the real world and proximity on the online social graph[16, 28, 20, 4]. Furthermore the small world effect is even greater in local communities [3], hence we question whether these effects continue to intensify in a more local level; the area of a city.

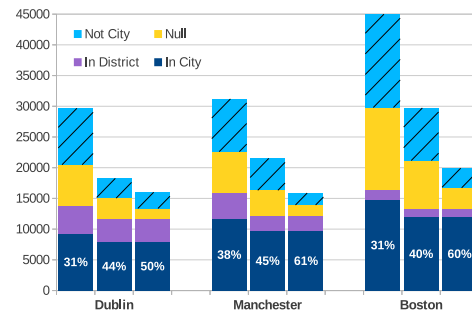
We collected tweets from a specific area using the Twitter streaming API, which provides, two types of tweets; a set which has exact geographical coordinates attached to each tweet and one which occurred within the bounds we set, but only has an approximate location. However, not all users who post a tweet from a city, live there. Some may have declared non-existing locations or nothing in their profiles, while others could be just visiting the city. In order to identify the location of a user we used a gazetteer provided by the GeoNames (www.geonames.org), from which we collected locations names from within our specified area, and checked if they matched a users' location. Understandably

our evaluation method is not perfect, since location names to not correspond to unique places on the map, however this is an insignificant percentage in our dataset.

In order to test whether locality in the real world is correlated to locality in the network, we apply a set of filters in our data. Initially we re-create a part of the social graph, only considering edges that are bidirectional. Most real world relationships by far exceed the energy of a mere follow back, and thus people that are connected in real life, are more likely to have a bidirectional edge than a unidirectional. The latter are more likely to occur when the two parties do not really know each other in the real world. Furthermore, we discarded very popular users which we defined as having more than  $f = 30000$  followers or friends, to avoid celebrities, who tend to be very central nodes, with very little location information.

Our first filter, was then to extract the largest connected component of the sample graph, formed from users who tweeted within  $C$ .

Secondly, we applied a measure in an attempt to remove users who are just visiting a city, we only kept users who tweeted more than  $n$  times in a period greater than  $h$  hours. For this experiment, we set  $n = 4$  tweets and  $h = 100$  hours.



**Figure 1: User Analytics for Each City.**

The first column of Figure 1 indicates statistics for users, for three different cities, namely *Dublin*, *Manchester*, and *Boston*. As expected in all three cases, more than 50% (the two top boxes) have the field empty or in a location which does not correspond to an area in the city. In contrast only a percentage in the order of 30% has a city level accuracy in their profiles, while the rest belong to a much larger region which includes out target city. We consider as regions, Ireland for Dublin, England for Manchester and the state of Massachusetts for Boston.

Interestingly, these numbers change significantly when we apply our first filter with all location categories being reduced, except for the percentage of people in the city. Since this is only a sample of the graph, being connected in this case, translates to stronger connectivity in the complete graph, which reaffirms that there is a correlation between locality in the real world and the online, and local communities seem to have a high clustering coefficient.

The third column illustrates the data after our third filter. Interestingly the users who pass this filter, and users in the largest connected component have many common nodes, who are mostly users who indeed live in the city. This fact holds true across all three cities which indicates that sample connectivity is a good pruning filter for local users.

Our intuition, based on our analysis, is that local communities on the real graph will form clusters with a high clustering coefficient in the social graph since there is a suggested correlation between location and friendship. From the data given above we can conclude that (i) the majority of users who live and tweet from the same city, are part of a strongly connected graph, with users that live there, and (ii) that connectivity from a sample is a good measure to prune users that do not belong in a city.

## 5. OUR APPROACH

Our approach formalizes the algorithm which considers the limited resources one has for the Twitter API. For this purpose, we built our methods considering our analysis of the social graph as well as the restrictions one can have in similar settings. Based on these observations, we built a method that is able to identify users from a specific city, by asking the minimum number of queries about the social graph.

### 5.1 Problem 1: User Discovery

For our first problem we use the **MaxEdge Algorithm**: From our set of tweets  $T$  that have geo-location within the area of a city  $C$ , we extract the users who created them. We then perform some *enhancement assessments*, based on our Analysis in 4.2 on them and consider this group of users to be our ground truth, from now on referred as *seed*. We then perform our graph discovery with the following algorithm: We create edges with weights from the *seed* to their friends, and create the rest of the known graph, referred to as *frontier*. In our method, initially all the edge weights are set to 1.0. Each of the nodes in the *frontier* has a score equal to the sum of the weights of the seed connected to it  $f_j = \sum w_{ij}$ . We then start to crawl by discovering the node with the maximum score. We query for his location and if this person is located in  $C$  we add him to our *seed*, query about his connections and update our social graph. We proceed in the same way until we exhaust our limit of  $k$ -queries.

Essentially our method is an enhanced first step of a BFS algorithm. Given that the graph in Twitter is mostly connected, a DFS is bound to escape the users living in area quite soon, simulating a random walk. Following this crawling method, at each step we create the most strongly connected graph possible, by maximizing the clustering coefficient of the seed. Given our analysis in Section 4.2, this increases the likelihood of finding users in the same city. In addition we minimize conductance, which is a measure indicating the quality of the community structure a part of a graph has [19]. In our case it is defined as the number of edges between the seed and the frontier over the number of edges inside the seed, which implies that a good community has low conductance. Our method's efficiency is based on the fact that it only queries the nodes that maximize the seed connectivity and minimize the conductance.

A representation of the graph formulated by the users we crawled, is in Figure 2. In this figure, the nodes on the left side, within the bounding box, represent the *seed* nodes denoted as  $s$ , which are users who tweeted from the city. As mentioned in section 4.2 a big part of these nodes are connected, while there are also smaller connected parts and nodes with few or no internal edges at all. The red nodes on the right side, represent our current *frontier*. These are

---

### Algorithm 1 MaxEdge Algorithm

---

**Input:** A set of users  $U$  who tweeted from within  $C$ ,  $k$

**Output:** New users in  $C$

```

seed ← ∅
for all <Ui in T> do
  if Assess(Ui) is true then
    seed += Ui
  updateFrontier(FriendsOfUi)
end if
end for
while k > 0 do
  NewUser ← MaxWeight(frontier)
  if NewUser in C then
    updateFrontier(FriendsOfNewUser)
  end if
  k ← k - 1
end while

```

---

nodes we have not queried yet, however they have at least one link with one node in the *seed*, hence we know of their existence, from the edge list of the  $s$ . Currently the node with the most edges to  $s$  has 4 edges, and if we discover and accept this node as being in  $C$ , there is a node with 3 edges who will then have 4 and will be our next query. Our method does not require us to maintain any internal edges within the seed, or nodes without external edges, thus limiting the space requirements and making it plausible to maintain in memory the information needed for a single city.

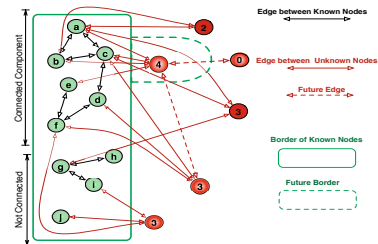


Figure 2: Seed and Frontier Example

### 5.2 Problem 2: Tweet Location Discovery

Our goal in this problem is to identify the geographical coordinates of a tweet. We evaluate two methodologies, one assuming we only have knowledge regarding the text of a tweet, and the other assuming we have information about the user who created it as well. For both methods, initially we segment the area, into squares of equal length  $a$ .

For the first approach, which we denote as **QL** we consider the Query Likelihood Model as defined in [17], since it provided the best results in [15]. However, instead of having different locations as different documents, we considered all the tweets that occurred in the same box as a document, and assign a query tweet to the box that has the maximum likelihood of having produced that tweet.

For our second method we exploit information about the users and his tweeting habits, to create the method **QLU**. In this method we assign to each square a probability that the user  $u$  tweeted from the box  $b$  as

$$p(User_u | Box_b) = \frac{|Tweets_u \in b|}{\sum Tweets_u}$$

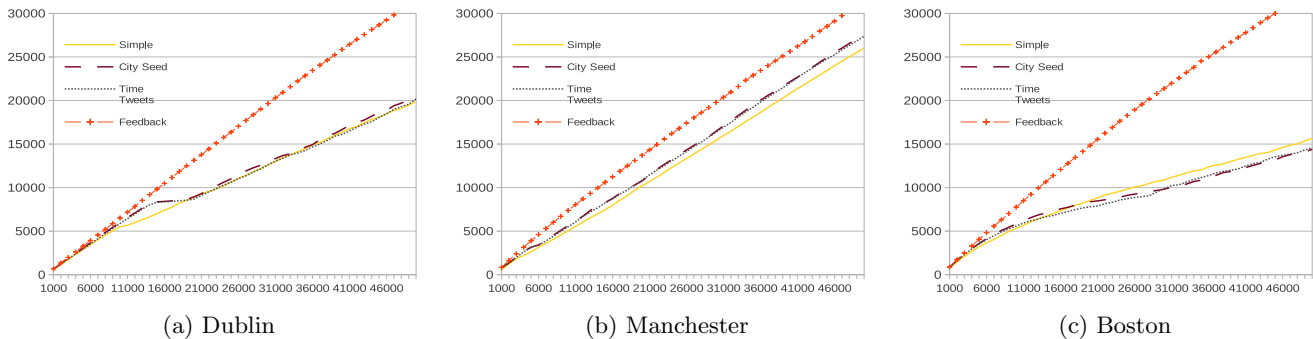


Figure 3: Users in  $C$  v Number of Queries in Twitter

and choose the box with the maximum probability  $p$ . If information regarding the user  $u$  is not available, we return to the **QL** method to locate it.

We consider this method as the next step of the User Discovery problem in a hierarchical model in order to identify the originating location of tweets. Adding geographical coordinates to a text works better if the possible area is limited, especially in the case of tweets, which contain mostly common words and very few location information.

## 6. EXPERIMENTS

### 6.1 Datasets

We run our experiments on geolocated tweets from 3 different cities. Our data were collected between April and August 2013, and their details are presented in Table 1. New Users, refers to the number of extra users we crawled with our method, that were from the same city. The Dublin All dataset, refers to all the tweets from users we were able to crawl that live in Dublin, which was used for the topic model experiments described in section 6.4

	Tweets	Users	Area	New Users
Dublin	1.9 M	43 k	1224 km <sup>2</sup>	179 k
Manchester	1.3 M	40 k	462 km <sup>2</sup>	70 k
Boston	1.5 M	55 k	1521 km <sup>2</sup>	73 k
Dublin All	71 M	220 k	1224 km <sup>2</sup>	-

Table 1: Dataset Details

### 6.2 User Discovery Results

In this experiment we measured how well our method, performs in discovering users from the a city, and illustrate our results for Boston, Dublin and Manchester, for  $k = 50000$ .

Initially we define as seed  $s$  all the users who tweeted from within  $C$  with an exact geo-location in their tweets. **Simple** refers to the algorithm described in 5.1 which takes all these users as its seed.

**City Seed** differentiates by considering as seed, the largest connected component of users who *declare* their in their profiles that they live in  $C$ .

**TimeTweets** ( $n, t$ ) considers as seed the city users who tweeted at least  $n = 4$  times with a difference of  $t = 100$  hours between the first and last tweet.

**Feedback** refers to a more complex version of our algorithm, which rewards the nodes that pointed to a correct user, and penalises those that pointed to an incorrect one. More

specifically, when updating the frontier, we increase or reduce the weights of the edges, of such nodes, by multiplying their current weight with a coefficient  $1 \pm c$ . For our experiments we set  $c = 0.01$ . This algorithm uses the same *seed* as **City Seed**.

#### 6.2.1 City Precision

Figure 3 illustrates that our method works well for the first 10 - 15 thousand queries, while filters in our initial seed outperform it slightly. After this point however, the part of the graph which we are trying to maximize its clustering coefficient, becomes too large and too connected with non-local users, and thus our method slowly begins to crawl non-city users. The exact number of dilution in the users, depends heavily on the size of the city, as smaller cities are more interconnected locally while larger cities tend to have more connections to outside communities [4].

This effect can be tackled by using our **Feedback** algorithm, which gave us the highest most stable precision in all cases, finishing with a precision of 62.9% for Dublin, 63.0% for Manchester and 65.6% for Boston. The results indicate that our intuition is correct, users in proximity can be discovered by maximizing connectivity, however after a number of queries, the seed becomes too diluted. **Feedback** essentially crawls the user, who is most likely to belong to a city, by the "majority vote" of all the users who live there.

Precision inevitably drops after a many queries, because the most certain users are already in the seed, and many users will have common connections from the outside world. The **Feedback** algorithm however, learns which users are more inclined to have connections with non local users and assigns less weight to their "votes".

#### 6.2.2 Region Precision

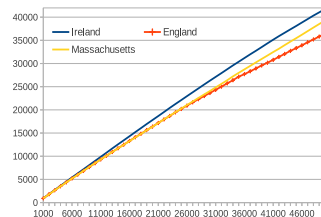


Figure 5: Users in Region v Number of Queries

Figure 5 represents the accuracy for regional results. The

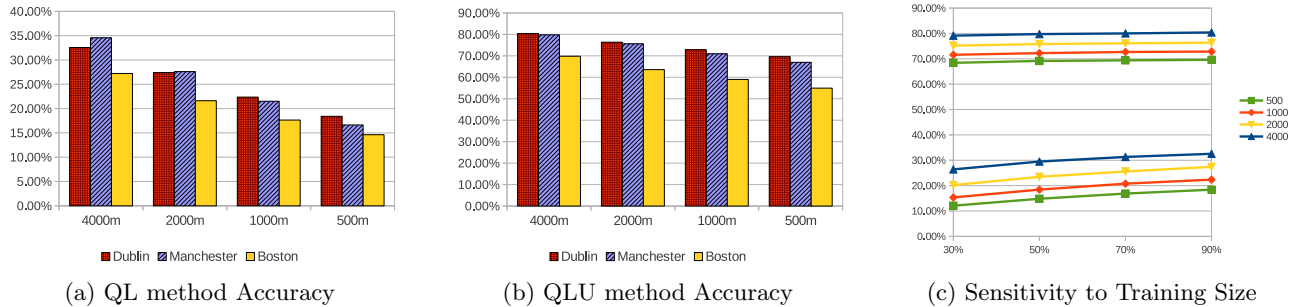


Figure 4: Size of Area - Boston: 39x39km, Manchester: 21x22km, Dublin: 34x36km

question we asked for this experiment, is whether focusing in a local area, will produce good precision, when accounting for users living in a greater region. We tested the **Feedback** algorithm, for a greater region surrounding our city. The precision here is higher, in the order of 80% however not proportionally to the increase in the real world size. This indicates that users are indeed more connected with others in the greater area of their location, however they are also connected globally.

### 6.3 Tweet Location Discovery Results

#### QL Method

This experiment evaluates how topical *tweets* are, and whether we can utilize their content in order to identify their exact geographical locations. For the **QL** method we preprocessed the data, by removing stopwords and performing a stemming function (through lucene K-Stemmer stemmer), which slightly improved our accuracy across all configurations. We tested for linear interpolation, however because of the small size of the query (maximum 140 characters) the best results were yielded when  $\lambda = 0$ . For each experiment we performed a 10-fold cross validation, by splitting the dataset in 10 subsets randomly, and for each experiment training with the 9 parts and tested with the other. The reported figure is the average precision, which had a trivial deviation.

Results of the **QL** method, for  $a = 4000$  reach a precision of 33% for Dublin, 34% for the city of Manchester and 27% for Boston, while they drop slightly as we reduce the size of the boxes. Figure 4(a) illustrates the accuracy for each city, for the tested granularities. We can observe, that although the number of possible boxes *quadruples* from each level of detail to the next, the accuracy of our method decreases in much slower fashion, even for the very specific granularity of 500m. It is also noticeable that Boston is always clearly lower than the other two cities, which can be attributed to the fact that the area we chose for Boston is almost four times the size of the area we chose for Manchester, as it is a much larger city. In addition, despite the fact that the area for Manchester is much smaller than the one for Dublin, they have very similar accuracies. This occurs due to the landscape of each city, since some of the area we selected for Dublin, inevitably contains a portion of sea, without any tweets. Table 2 contains more detailed information about the total number of tweets, the number of empty and reported squares, as well as the average Error Distance in kilometres for each city; for  $a = 1000m$ . Interestingly,

the average distance is only slightly reduced, when precision is increased which indicates that our method often chooses neighbouring boxes for smaller granularities. Our method reports the majority of squares, which indicates that it is not affected by the skewed distribution. However this biased distribution cannot be exploited to locate tweets. In order to illustrate that, we created a method that assigns a tweet to a location with a probability proportional to the number of tweets in that box, which yielded an accuracy of less than 1%.

	Grid Size	Tweets	Empty	Reported	Error
Dublin	34 x 36	1099904	309	868	11.604
Manchester	21 x 22	798779	20	442	8.301
Boston	39 x 39	1014232	117	1521	14.822

Table 2: Details for QL experiments

#### 6.3.1 QLU Method

In this experiment we evaluate whether *users* are topical, and tend to tweet from the same places, when they do so by their phones. Figure 4(b) presents the same information as the previous section, for the method **QLU** which yielded a precision of 80% for boxes of side 4000m in Dublin, which was reduced only by 10% when reducing the size of the reported area to 1/64 of the original. This result indicates that users, across cities, tweet in a spatial routine pattern, with a very big portion of their tweets from the same specified area.

Table 4(c) shows the sensitivity of our method when using various percentages of the dataset as training. The upper lines represent the accuracy **QLU** method while the bottom ones represent **QL**. The latter is sensitive for tweet number, however **QLU** is not, which indicates the strength of people’s habits to tweet from nearby locations.

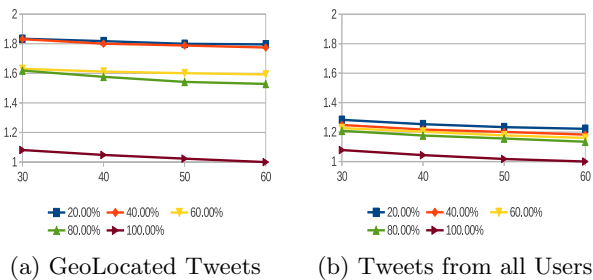
### 6.4 Evaluation through Topic Models

This experiment illustrates how well we can summarize the information from all the tweets we crawled, as well as the effectiveness and necessity of our methods since we can extract more meaningful and accurate topic models through LDA [5]. Summarizing tweets through topics is important, especially in the case of emergency identification. Modelling the topics of discussion in a city, can aid towards defining which are usual topics of discussion in a city, and thus the ability to spot abnormal ones. Furthermore it can be used to identify the rate with which people change topics of dis-



cussion, and identify a city-wide emergency in case of an abrupt change, directions we are eager to investigate in the future.

For the city of Dublin, we initially used the geo-located tweets we received from Twitter, and then those we were able to download from the new users we discovered with our method. Assuming that one wants to know what the major topics of discussion in a city  $C$  are, during a time interval  $t$ , we identify the difference between the two datasets. As a measure of performance of LDA, we use perplexity as defined in [27]. Perplexity essentially tells us how well our probability distributions over the topics, represent the testing set. Our results indicate that the set of GeoLocated Tweets is much more perplexed than the complete dataset, for various training percentages, across different total number of topics. Except for the case of the complete dataset being a training set, the rest of the samples are non inclusive.



**Figure 6: Perplexity v Number of Topics, for various percentages of the Dataset (lower is better)**

Figure 6(a) shows the perplexity of the geo-Tweets when divided with the perplexity of the complete set and most topics which is the lowest. We can observe that the perplexity drops significantly when using a bigger dataset, which is of-course expected. However in the second case (Figure 6(b)), which is the dataset with all the tweets, the perplexity is at all levels much lower and less sensitive. This indicates two things: Firstly, LDA creates better topic models given our more holistic dataset, and secondly that even with a small percentage of that data, we are able to create and retrieve a more coherent picture about the topics discussed in a city than just with geo-located tweets. In other words, the tweets which are geo-located are not enough to accurately depict what the topics of conversation in a city are.

We also performed a qualitative test, by manually labelling the topics and determining how many can potentially be annotated with a coherent topic of discussion. We labelled the 20% and 100% of both datasets. After considering the results from the largest dataset as ground truth, since it contains all other datasets, we evaluated how many topics from each dataset are relevant to the this. The results are in Table 3 . We present some indicative topics and annotations for each case in Table 4.

	Relevant	Annotated
20% Geo	10	32
100% Geo	18	37
20% All	27	42
100% All	45	45

**Table 3: Relevant and Annotated Topics**

## 7. CONCLUSIONS

Identifying information for a specific location is an important problem. In this regard we analysed the structure of social networks for users that live in proximity and concluded that there is a correlation between strong connectivity in the social graph and proximity in the real world. We created a method that captures the dynamic relations on such a graph, and can locate users who live in a specific area, optimized for minimum number of queries in the graph. Furthermore we created a method for precise geo-location of tweets within a city with high accuracy and provided extensive experimentation on a real social network, regarding the effectiveness of our method as well as the quantitative and qualitative benefit from the newly found data.

## 8. ACKNOWLEDGMENTS

This work has been co-financed by EU and Greek National funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Programs: THALIS - GeomComp, THALIS - DISFER, ARISTEIA - MMD" and the EU FP7 funded project INSIGHT (www.insight-ict.eu).

The authors would like to thank Myrto Vlazaki and Ioannis Katakis for their their valuable insights and recommendations

## 9. REFERENCES

- [1] A. Agarwal, K. Hosanagar, and M. D. Smith. Location, Location, Location: An Analysis of Profitability of Position in Online Advertising Markets. *Journal of Marketing Research (JMR)*, 48(6):1057–1073, 2011.
- [2] A. Ahmed, L. Hong, and A. J. Smola. Hierarchical geographical modeling of user locations from social media posts. In *Proceedings of the 22nd international conference on World Wide Web, WWW '13*, pages 25–36, 2013.
- [3] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna. Four degrees of separation. In *WebSci*, pages 33–42, 2012.
- [4] L. Backstrom, E. Sun, and C. Marlow. Find me if you can: improving geographical prediction with social and spatial proximity. In *Proceedings of the 19th international conference on World wide web, WWW '10*, pages 61–70, 2010.
- [5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, Mar. 2003.
- [6] H. S. Burton, W. K. Tanner, G. C. Giraud-Carrier, H. J. West, and D. M. Barnes. Right time, right place health communication on twitter: Value and accuracy of location information. *J Med Internet Res*, 14(6):156, November 2012.
- [7] Z. Cheng, J. Caverlee, and K. Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management, CIKM '10*, pages 759–768, 2010.
- [8] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social

20% Geo	abortion	ireland women abortion vote france vinb law issue politics country bill pro europe reading point government tax people bank
20% Geo	tourism	dublin ireland bar park photo guin dub street post tha cliath howth aerfort st bhaile square hotel temple green
20% Geo	football	united mufc great castle de enjoy day pal weekend love mate massive yea matchday stanakatic casket hun
20% Geo	rugby	lion leinster rugby ihl fitness lions coybib gatland trx bod fitfam pm ym seaofre final match anglotape fella heinekencup
20% Geo	?	donegal christina tom bloomsday roscomon fe casa kenny patrick theapprentice hill floor princess rackard ding alley min
100% Geo	places	dublin ireland bar guin dub tha cliath aerfort bhaile pic hotel airport temple storehouse pub st pint trinity college
100% Geo	football: uk	player united season game play arsenal football sign team haha league fan goal win suarez man mate mufc chelsea
100% Geo	music show	check play rt music gig album video film listen show eventsindublin live song awesome watch band festival game cat
100% Geo	rugby	lion game great win match play final gaa leinster team rugby golf ireland today wimbledon player murray congrats dub
100% Geo	?	watch man life show call god car made hit kid years laugh mr tom men face jesus dream break
20% all	irish jobs	business ireland job dublin social jobfair marketing digital media hire great tip irishjob startup network tech sales online company
20% all	Politics:EC	ireland bank tax eu people europe pay report government uk protest minister year state home court news service job
20% all	food	food restaurant wine lunch taste special beer coffee free cocktail dinner menu delicious chef recipe bar eat lovely yum
20% all	abortion	abortion vinb life women ireland bill vote seanad pro people baby politics law party dail fg debate prolife labour
20% all	?	ur ya ye im goin yea dont wat tho pal ha gettin il wit ill ah nite jus bout
100% all	Politics: EC	ireland bank tax news eu britain report uk government police protest europe belfast human gold syria attack year minister
100% all	research	ireland great today health support student research eu event conference school day people children education week work launch europe
100% all	abortion	abortion ireland vinb women vote life people bill seanad law party support politics pro dail debate woman gay marriage
100% all	irish jobs	job dublin ireland business jobfair irishbizparty manager hire sales recruit cork company bizhub engineer developer senior service client
100% all	?	ur ya ye ha goin haha wat tho gettin il wit im dat de jus day yea nite

Table 4: Topics from the LDA for each dataset, and their annotation

- networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '11, pages 1082–1090, 2011.
- [9] J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing. A latent variable model for geographic lexical variation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1277–1287, 2010.
- [10] G. Fuchs, G. Andrienko, N. Andrienko, and P. Jankowski. Extracting personal behavioral patterns from geo-referenced tweets. 2013.
- [11] B. Hecht, L. Hong, B. Suh, and E. H. Chi. Tweets from justin bieber's heart: the dynamics of the location field in user profiles. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 237–246, 2011.
- [12] R. Holt. Twitter in numbers, Mar. 2013. <http://www.telegraph.co.uk>.
- [13] L. Hong, A. Ahmed, S. Gurumurthy, A. J. Smola, and K. Tsioutsoulouklis. Discovering geographical topics in the twitter stream. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 769–778, New York, NY, USA, 2012. ACM.
- [14] Y. Ikawa, M. Enoki, and M. Tatsubori. Location inference using microblog messages. In *Proceedings of the 21st international conference companion on World Wide Web*, WWW '12 Companion, pages 687–690, 2012.
- [15] S. Kinsella, V. Murdock, and N. O'Hare. "i'm eating a sandwich in glasgow": modeling locations with tweets. In *Proceedings of the 3rd international workshop on Search and mining user-generated contents*, SMUC '11, pages 61–68, 2011.
- [16] H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM.
- [17] J. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR*, SIGIR '01, pages 111–119, 2001.
- [18] T. Lappas, M. R. Vieira, D. Gunopulos, and V. J. Tsotras. On the spatiotemporal burstiness of terms. *Proc. VLDB Endow.*, 5(9):836–847, May 2012.
- [19] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704, 2008.
- [20] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 1023–1031, 2012.
- [21] W. Li, P. Serdyukov, A. P. de Vries, C. Eickhoff, and M. Larson. The where in the tweet. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 2473–2476, 2011.
- [22] J. Mahmud, J. Nichols, and C. Drews. Where is this tweet from? inferring home locations of twitter users. In J. G. Breslin, N. B. Ellison, J. G. Shanahan, and Z. Tufekci, editors, *ICWSM*. The AAAI Press, 2012.
- [23] S. Murray. Twitter breaks news of whitney houston death 27 minutes before press, Feb. 2012. <http://mashable.com>.
- [24] K. Ren, S. Zhang, and H. Lin. Where are you settling down: Geo-locating twitter users based on tweets and social networks. In *Information Retrieval Technology*, volume 7675 of *Lecture Notes in Computer Science*, pages 150–161. 2012.
- [25] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 851–860, 2010.
- [26] G. Valkanas and D. Gunopulos. Location extraction from social networks with commodity software and online data. In *ICDM Workshops*, pages 827–834, 2012.
- [27] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. Evaluation methods for topic models. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1105–1112, 2009.
- [28] S. Yardi and D. Boyd. Tweeting from the town square: Measuring geographic local networks. In *ICWSM*, 2010.



# Efficient Dissemination of Emergency Information using a Social Network

Iouliana Litou  
Department of Informatics  
Athens University of  
Economics and Business,  
Athens, Greece  
litou@aub.gr

Ioannis Boutsis  
Department of Informatics  
Athens University of  
Economics and Business,  
Athens, Greece  
mpoutsis@aub.gr

Vana Kalogeraki  
Department of Informatics  
Athens University of  
Economics and Business,  
Athens, Greece  
vana@aub.gr

## ABSTRACT

In the recent years social networks have undergone explosive growth. They have been used as major tools for the spread of information, ideas and notifications among the members of the network. In this paper we aim at exploiting this new communication channel for emergency notification, to deliver emergency information to all appropriate recipients. We develop ESCAPE, our system for efficient dissemination of emergency information in social networks. We propose an approach that investigates the interactions and relationships established between the members of the social group, and develops a scalable dissemination mechanism that selects the most efficient routes to maximize the information reach. Our experimental results illustrate the feasibility and performance of our approach.

## Categories and Subject Descriptors

C.2.4 [Distributed Systems]

## Keywords

Distributed Systems, Social Networks, Information Dissemination

## 1. INTRODUCTION

In the recent years social networks such as Facebook, Twitter and Google+ have undergone an explosive growth, enumerating large numbers of subscribers. For example, Facebook counts over 900 million active users, followed by Twitter with over 550 million users and Google+ with over 170 million users<sup>1</sup>. They have been used as major tools for the spread of information, ideas and notifications among the members of the network. Recent studies reveal that social networks can be used efficiently not only for “viral marketing” [17] to promote new products to targeted sets of users

<sup>1</sup><http://www.go-gulf.com/social-networking-users.jpg>

which further propagate the products through the “word-of-mouth” effect to reach a larger audience, but also for discovering emergent topics[1], emergency alerting, management and public safety [16].

Consider for example the emergency event of an earthquake. People in the vicinity of earthquakes are sharing anecdotal information that earthquake alerts lagged behind firsthand notification sent through Twitter, a popular Internet-based service for sending and receiving short text messages[2, 9]. The study reveals that depending on the size and location of the earthquake, scientific alerts can take between two to twenty minutes to publish, while using Twitter’s notification capabilities people were notified about the occurrence of the earthquakes shaking within seconds of their occurrences.

Combining geographic coordinates with social networks, enables social networks to interact with users relative to their locations, or connect users with local events, places or groups that match their interests. This is becoming increasingly popular in several geosocial applications such as Facebook Places<sup>2</sup> and Foursquare<sup>3</sup>, where users are allowed to share their geographic locations as well as make and receive recommendations for a set of venues. In an emergency scenario, such geosocial networks contribute not only to develop a collective situational awareness about the event, but also allow users to coordinate around geotag information related to hazards and disaster aid activities.

Thus, social networks (*i.e.*, Twitter<sup>4</sup>, Facebook<sup>5</sup>, LinkedIn<sup>6</sup>) are opening new avenues for massive emergency notification due to their ability to (1) reach millions of social network users, especially family and friends, (2) become alternative communication mediums when wireless and telecommunication networks are congested during emergencies, and (3) provide cost-effective solutions as they have the ability to reach massive amounts of users without added infrastructure costs.

However, adopting social networks as an effective communication medium for emergency alerting raises considerable challenges in the level of availability and responsiveness ex-

(c) 2014, Copyright is with the authors.

Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

<sup>2</sup><https://www.facebook.com/about/location>

<sup>3</sup><http://www.foursquare.com>

<sup>4</sup><https://twitter.com/>

<sup>5</sup><https://www.facebook.com/>

<sup>6</sup><http://www.linkedin.com/>

pected from these infrastructures in delivering notifications to reach all recipients interested in receiving this information (these can be people located in the area of the event *i.e.*, students in a campus, as well as their relatives and friends).

In this paper we illustrate the problem of how to leverage the social network for efficient dissemination of emergency information. Our objective is stated as follows: Given a social network comprising a number of users, the social relationships of the users and the set of recipients, our goal is to select an appropriate subset of the users to propagate the emergency information such that (1) the expected spread of information is maximized among interested users, (2) costs are considered. Cost is defined as the amount of messages that need to be exchanged among users. Thus, it could be translated as either monetary cost (for an SMS) or resource allocation cost.

We approach the problem by following discrete procedures where user profiles are built, social relationships are inferred and dissemination paths among the nodes of the social network are computed and during the occurrence of an emergency event, a small number of seed nodes is selected to efficiently disseminate the emergency information to all interested recipients during the event.

Current influence maximization approaches are not adequate to solve these problems. The problem of maximizing the spread of influence in social graphs has been addressed in [8, 19, 24, 13], but none of these works has study the problem in the context of emergency notification. Furthermore, they aim at maximizing the influence in the entire network rather than identifying and informing an appropriate subset of nodes that would be most interested to the event.

Emergency response outside social networks has also been studied. The use of geographical notification systems has been considered in [14]. The system described is meant primary for constructing overlays that support location-based regional multicasting where they also consider issues of providing reliable storage of social information events under extreme regional conditions. Traditional approaches such as multicast [21, 11] and publish/subscribe systems [18] are not appropriate for our setting since they will inform only subscribed users, while we need to alert all users associated with the emergency event. In our system, we consider users are already subscribed to the network. Moreover, the set of users to be informed by our system is not determined based only on locational criteria, but also on relationship criteria, so its not considered to be a strictly location-based approach.

On the other hand, approaches like flooding and gossiping [4, 6, 7] will inform most of the users interested in the event, but they will also produce a large amount of spamming to the other users, thus adding extra cost to the network.

Our paper makes the following contributions:

- We present ESCAPE (Efficient diSsemination using soCiAl graPh for Emergency response), a system that solves the problem of efficient dissemination of emergency information in social networks. We show that

the problem of selecting an appropriate influential set of individuals to maximize the spread of information is NP-hard and provide a greedy algorithm to solve it. We do not aim at spread maximization as previous works, but rather at reachability maximization of a subset of users with the least cost.

- We perform experiments to validate our approach. Our experimental results illustrate that our approach is practical and effectively addresses the problem of informing the maximum amount of users with the least messages when an emergency event occurs, and outperforms its competitors.

## 2. BACKGROUND

Kempe et al.[13] were the first to propose cascade models in Social Networks. They define two models describing the way influence is propagated in Social networks, namely the Independent Cascade Model and the Linear Threshold Model.

Both models require that a weighted graph representing the social network is given. In the graph, nodes represent users, edges represent influence flow between users and weights represent the probability that the influence propagation is successful among nodes connected with that edge.

In the Linear Threshold Model, aside from the weights to the edges, a threshold is also associated with each user. That threshold expresses the susceptibility of a user to the influence. Nodes that are already influenced are referred as active nodes and the remaining as inactive. A node in the Linear Threshold Model is considered to be activated when the sum of the edges of its currently active neighbors reaches the threshold.

Unlike the Linear Threshold Model, in the Independent Cascade Model no thresholds are considered. The cascade is progressing in steps and at each step, currently active nodes have the chance to influence their neighbors. Nodes have only a single chance of activating their neighbors, and the probability that they succeed is defined by the weight of the edge.

In this work we consider the Independent Cascade Model to be more appropriate of describing the way the information is spread.

## 3. PROBLEM DEFINITION

We now provide a formulation of the problem and prove its NP-completeness. Consider a social graph  $G = (V, E, W)$ , where each vertex  $u \in V$  represents a user, each edge  $e_{uv} \in E$  denotes a social relation between a pair of users  $(u, v)$ , and  $w_{uv}$  corresponds to the strength of the relationship between the users. The relations between the users are assumed to take place over the lifespan of the network (an edge occurs between a pair of users if and only if the two users are connected socially in some manner). Given a social graph  $G = (V, E, W)$ , a subset of the vertices  $S \subset V$ , and a positive integer  $k < |V|$ , our goal is to find a seed set  $M \subset V$ , such that the expected number of nodes in  $S$  informed by  $M$  is maximized, and  $|M| \leq k$ .

Thus, the problem to be solved is how to maximize the

amount of nodes  $n \in S$  that will be informed given a maximum amount of  $k$  seeds that can be used. Note however that not all nodes are constantly connected to the system. Thus, we aim at maximizing the information spread by selecting at most  $k$  vertices from  $V$  to efficiently disseminate the messages, under the condition that these nodes are connected.

Our problem differs from traditional influence maximization problems, such as the Independent Cascade (IC) Model [13]. The difference is that our goal is to inform a subset of nodes,  $S \subset V$ , referred as *interested nodes*, which are closely affected by the event. Thus, we aim at maximizing the number of nodes  $n \in S$  that will be informed, rather than informing all the nodes in the graph. The key challenge here is that the reachability of the nodes, in terms of physical connectivity, introduces constraints on the availability of the nodes in the graph, since there may only be a subset  $R$  of nodes that can be reached. Thus, not all nodes of the network are candidate seeds.

Similarly to the IC model, whenever a node is informed, there exists only one chance that this node forwards the message to its neighbors. The problem as stated above is NP-complete. The reduction from Hitting-Set to this problem is quite trivial. The Hitting-Set problem is defined as described below.

*Hitting-Set:* Given a set  $A = a_1, \dots, a_n$  and a collection  $B_1, \dots, B_m$  of subsets of  $A$  and a number  $k$ . There exists a Hitting-Set  $H \subseteq A$  of size  $k$  such that  $H \cap B_i \neq \emptyset$ ,  $1 \leq i \leq m$ .

**Proof: Reduction from Hitting-Set problem.** If we consider the original set  $V$  of nodes and subsets of  $V$   $B_1, B_2, \dots, B_m$  constructed in a way that if one node in  $B_i$  is informed then all others in  $B_i$  are informed too, which is an assumption that makes the problem easier, then we need to find a set  $H \subseteq V$  of size  $k$  such that  $H \cap B_i \neq \emptyset$ ,  $1 \leq i \leq m$ . Without considering the assumption  $H \subseteq R$  the problem is NP-Complete. The restriction just makes it harder.

Since the problem is NP-Complete we develop an approximation algorithm to solve the problem efficiently.

### 3.1 Overview of the ESCAPE System

In this section we provide an overview of our ESCAPE (Efficient diSsemination using soCiAl graPh for Emergency response) system. In order to achieve maximum reachability of interested users, the system implements the following procedures: i) Profiling of users based on past actions, ii) Social Strength assignment, denoted as the weights of the edges among users in the social graph and iii) a Dynamic Notification of a subset of users (referred as seeds) to initially receive the information when an event occurs, and further propagate it.

As users connect to the social network, it is possible to extract user information by exploiting the networks created and the messages exchanged among the users. The Profiling procedure (further discussed in 4.1) is responsible for building user profiles and maintain user statistics. It uses the raw data of the user interactions to generate a list  $I_u$  for each

user  $u \in V$ , that contains the interactions of user  $u$  with any other user in the network. The weights of the social graph  $G(V, E, W)$  are inferred based on the information extracted by the Profiling procedure and it characterizes the social relationships among the users based on past interactions, thus it denotes the Social Strength among a pair of user in the graph.

Finally, when an event occurs the Dynamic Notification procedure is triggered. Information computed and maintained by the Social Graph that is formed among users is utilized to identify the initial receivers of the message so as to maximize the spread of information to the interested users, while using the least amount of messages.

The overall architecture and the interaction between different procedures is depicted in Figure 1 and the corresponding functionalities are described in the following sections.

## 4. EFFICIENT DISSEMINATION OF EMERGENCY INFORMATION

In this section we start by describing the metrics we use to identify and characterize the social relationships among the users in the social graph and then we present our dynamic notification algorithm that aims at selecting a subset of the users to propagate the information.

### 4.1 Profiling

To construct the social graph, the first step is to identify and characterize the social relationships through the messages exchanged among the users. We build user profiles for each user  $u \in V$ . Each user  $u$  is associated with a unique id, this for example might be the user's id in the social network, and is used each time the user logs into the system.

Whenever user  $u$  sends a message to a user  $v$ , the list of interactions  $I_u$  of user  $u$  is updated. The form of the tuples in the  $I_u$  list is:  $\langle v, m_v, timestamp_v \rangle$ , where  $v$  is the id of the receiver of the message  $m_v$ ,  $timestamp_v$  denotes the unix timestamp when the message was sent.

The data retrieved from the user profiles is used for the Social Graph construction  $G(V, E, W)$ . This is represented as a directed weighted graph. The graph is not required to be updated in a continuous manner, but in discrete time intervals. Each user  $u$  in the Social Network forms a node in the graph. For each node  $v$  in the  $I_u$  list of user  $u$ , an edge is instantiated in the graph between  $u$  and  $v$ . We associate a weight with each edge in the graph to represent the "strength of the relationship" among the users in the social network. The weight takes values in the range between  $(0, 1]$ , where a value of 1 denotes a strong relationship between the users. The weights  $w_{uv}$  are assigned according to one of the metrics described below. We note that there are several mechanisms for assigning weights to edges as stated in the related work section [10, 22, 25]. Although in our work we focus on the metrics described below, the model is generic enough and can be extended so that several metrics can be considered.

To compute the weights of the edges we consider a simple metric, that represent the "social strength" of the relationship between two users:

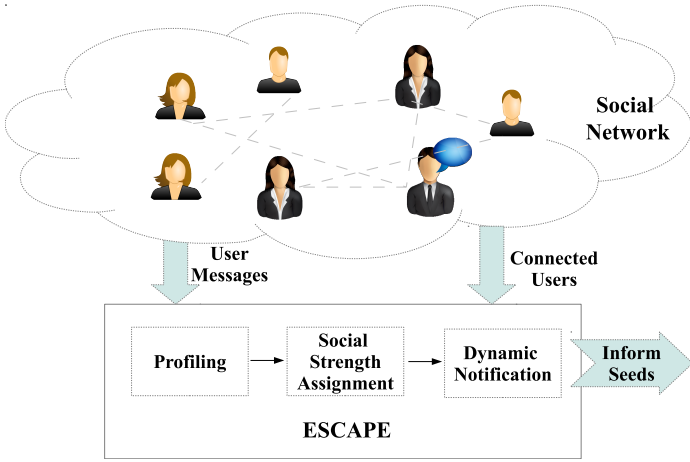


Figure 1: The ESCAPE Architecture.

- *Frequency of communication (FC)*. Usually people that are strongly connected to each other, communicate more often. Thus, the frequency of the communication captures the strength of the relationship between the users. The frequency is not defined as an absolute value for all users (i.e. twice a day) as some users are more sociable than others, and that should be taken into account. So, the frequency of communication between a pair of users  $(u, v)$  is defined as:

$$f_{uv} = |m_v| \in I_u / \sum_{i \in I_u} m_i \quad (1)$$

Thus,  $f_{uv}$ , for user  $u$ , denotes the amount of messages exchanged with user  $(v)$ , denoted as  $m_v$ , out of the total messages that user  $u$  has sent to every other user  $i$ ,  $m_i$ . In the case of the Twitter social network, the strength of the user relationships is perceived through the tweets exchanged. It is important to note that  $f_{uv}$  differs from  $f_{vu}$ , since Twitter presents large asymmetry in the relations due to broadcasters and miscreants [15].

- *Regularity of communication (RC)*. It is known that people having strong social bonds may not necessarily communicate very often, but they may be communicating at regular time intervals. So regularity of communication is also considered as an important factor in calculating the strength of a relationship. The regularity metric is defined as:

$$f_{uv} = 1 / \log(d_{uv} + 1) \quad (2)$$

where  $d_{uv}$  represents an average time lapse (e.g. days) between user communication. For instance, if user  $u$  interacts at each time lapse (e.g. daily) with user  $v$  then  $d_{uv}$  equals 1. The regularity is time-window based and considers the regularity of communication within this time window.

We compute the normalized weight so that it is insensitive to the user's special characteristics and does not depend on the set of data measured for a specific user as:

$$w_{uv} = f_{uv} / \max\{f_{uv'} : v' \in \text{neighbor}(u)\} \quad (3)$$

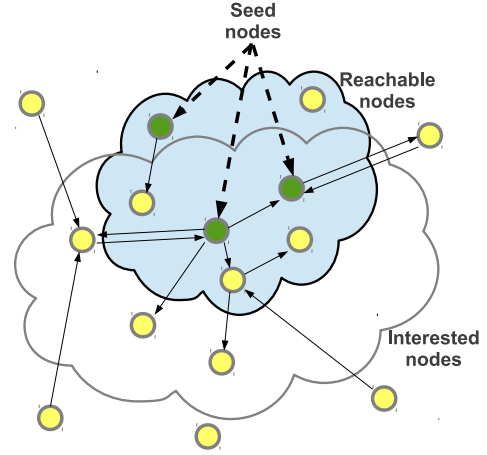


Figure 2: The Node Types.

Note, though, that the above mechanisms aim at deriving the strength of a relationship among users automatically, without user involvement. However, in some cases, it is desirable that users are given the opportunity to define their own set of individuals to be informed in cases of emergency. In this case, the user defines the list of emergency contacts and the corresponding weights of those edges equal 1, regardless of the metric.

For our experiments we use the FC metric for characterizing the weights among users in the network, since it is more appropriate for our dataset, as RC may require data collected for a longer time to appropriately and stably characterize the strength.

## 4.2 Dynamic Notification

Whenever an event occurs we identify the following roles among the users (shown in Figure 2): (a) **Interested nodes** are all nodes that are interested in the occurrence of the event. These are nodes that are more important in information spreading process and the ones we aim to reach. They are subset of the original graph and we represent them as the nodes belonging to the white cloud in the figure. They represent users within the area of the event or users related to them or to the area in someway, but are not physically there. (b) **Reachable nodes** are the nodes that can be reached after the occurrence of the event, i.e. information can be directly sent to them and they can be either interested or not. These nodes are the ones that belong to the blue cloud. (c) **Seed nodes** are the nodes to which information is initially sent, so they are the nodes that we aim at identifying to initiate the propagation process. These nodes are subset of the reachable nodes and we illustrate them as green nodes.

The basic functions of the Dynamic Notification process is to determine the users interested in the event, and select an initial set of nodes (*seeds*) that will be informed for the event and initialize the propagation process to reach interested users. That is accomplished using a greedy approximation algorithm in order to maximize the finally reached nodes given the size of seed nodes. It is noted that seeds are nodes that can be accessed during the occurrence of the

event, thus consists of the reachable nodes in figure 2.

#### 4.2.1 Seed Selection

When an event occurs, the seed selection process is triggered. The first step of the process is to determine the interested nodes. These can be determined in various ways based on geosocial criteria. They could be: (i) users who are physically located to the place of the event, (ii) related to users (*i.e.* relatives), (iii) related to the place of the event (*i.e.* students). The next step is to determine the reachable nodes in the network *i.e.* nodes that are connected and are accessible, that is they are connected in the social network immediately after the event (*e.g.* these can be obtained when a user logs into the network). Let  $R$  be the set of reachable nodes and  $S$  be the set of interested to the event users. For nodes in  $R$  we determine the paths that start at node  $u \in R$  and terminate at node  $v \in S$ . For those paths the probability  $p(u,v)$  that a message initialized by  $u$  traverses the path and reaches  $v$  is calculated. The probability that the path is traversed can be calculated in various ways. For our experiments we define the probability as the product of the weights of all edges that must be traversed from  $u$  in order to reach  $v$ . After probabilities are calculated the greedy step for the seed selection process follows.

*Greedy Node Selection:* Consider a set of seeds  $M$ , a set of nodes  $A$  that we expect that will be informed by  $M$ , thus  $M \subseteq A$ , the set of interested nodes  $S$ , the set of reachable nodes  $R$ , and a candidate seed  $u \in R$  to be added in  $M$ . The greedy step for the selection of  $u$  is:

$$M \cup u \text{ s.t. } \sigma(u) = \max\{\sigma(v) : \forall v \in R \setminus A\} \quad (4)$$

$$\text{where } \sigma(u) = \frac{(\sum_{v \in S \setminus A} p(u,v))^2}{|S \setminus A|}$$

Intuitively,  $\sigma(u)$  computes the number of interested nodes  $v \in S$  informed when  $u$  is selected as the root of the dissemination process, while taking into account the average probability that those nodes are effectively informed. Nodes that are possibly informed by previous seeds selected are not computed to the efficiency of  $u$ . Thus, the algorithm may produce a seed set that has multiple seeds reaching the same node in  $S$ , *i.e.* consider nodes in  $u_1, u_2 \in M$  that both have paths to node  $v \in S$ . This case is not undesirable since the chance of  $v$  being informed is increased. Possibly informed nodes are computed using Monte-Carlo simulation while considering as seed only the latest node added in the seed set.

The above function for seed selection can be proved to be monotonous and submodular, and thus approximates the best solution.

#### 4.2.2 Dissemination of Information

Assuming a set of seed nodes  $M$ , our model propagates the emergency information in a number of steps. Let  $A_t$  be the set of nodes informed at step  $t$ . At the beginning, the seed nodes are the only ones informed and thus  $A_0 = M$ . At step  $t+1$ , every node  $u \in A_t$  is able to inform each of its currently uninformed neighbors  $v$  and the probability that  $u$  informs  $v$  is given by the probability denoted by the weight of the

edge  $(u,v)$ ,  $w_{uv}$ . Each node has a single chance of informing its currently uninformed neighbors about the event.

The above described procedures are summarized in algorithm 1.

---

#### Algorithm 1 Efficient Dissemination

---

```

 $S_1 \rightarrow$  Directly connected to the event users
 $S_2 \rightarrow$  Compute Indirectly Connected to the Event ( $S_1$ )
 $S \rightarrow S_1 \cup S_2$ 
 $R \rightarrow$  get reachable nodes( $G$ )
 $Seeds \rightarrow \emptyset$  //nodes that will act as propagation starters (Subset of  $R$ )
 $A \rightarrow \emptyset$  //nodes possibly informed by previously selected seeds
while ( $|Seeds| < k$ ) do
     $newseed \rightarrow$  get most effective node( $G, R, S, A$ )
     $A \rightarrow$  add possibly informed users( $newseed$ )
return Seeds

```

---

## 5. EXPERIMENTAL EVALUATION

### 5.1 Experimental Setup

We have implemented our ESCAPE system and tested it with a real-world dataset.

Our Twitter dataset is composed of 513,449 tweets posted by 175,974 unique users in the city of Dublin for a four-month period (Dec 2012 to Mar 2013). We collected tweets using the Streaming API <sup>7</sup> of Twitter, where we applied a filter so that only tweets geographically located in Dublin are extracted. The filtering is based on the "Location" field set by the user and is either expressed as GPS coordinates or as part of the text of the tweet[23]. Tweets that were posted and had GPS location(Latitude, Longitude) provided by the devices' GPS sensor are also extracted. After tweets related to Dublin are extracted, users that posted tweets are gathered and by using the REST API <sup>8</sup> and the user ids, all tweets for each user are requested. A tweet has the following structure <Tweet ID, User ID, UTC/GMT timestamp, Latitude, Longitude, ID of tweet replying, ID of user replying, Number of retweets, Source (iPad, Android), Text>. From this structure, the User ID is used to obtain the screen name of the user, the UTC/GMT timestamps are used to compute the regularity metric and the Text is used to extract the users mentions with the '@' symbol. No further information related to the users is used. We do not consider any anonymization issue [5] since we assume that the authority that executes the system can be trusted (*e.g.*, the case of a campus social network). Twitter users can be classified into three major clusters as previously shown in [15]. These clusters are the broadcasters, the acquaintances and miscreants. Due to this asymmetry presented between users the graph that is formed is a directed weighted graph.

The experimental evaluation focuses on the following parameters: (i) Number of Informed Nodes from the users that are interested to be alerted for the event and (ii) Performance under different amount of Reachable users.

<sup>7</sup><https://dev.twitter.com/docs/streaming-apis>

<sup>8</sup><https://dev.twitter.com/docs/api>

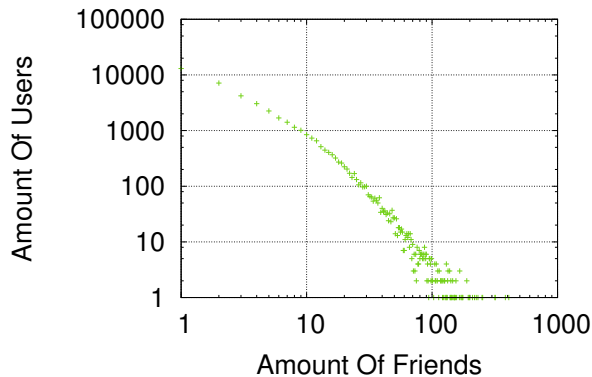


Figure 3: Friends of Users

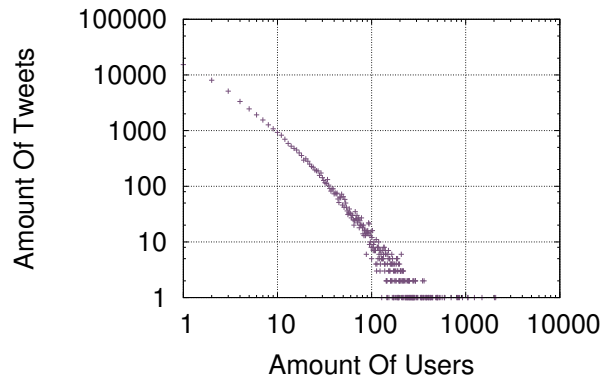


Figure 4: Tweets of Users

### 5.1.1 IRIE Overview

Jung et al. proposed IRIE[12], an algorithm that incorporates Influence Ranking and Influence Estimation for the influence maximization problem in the IC and IC-N (negative opinion propagation) model, which is proved to run faster than previous algorithms aiming to solve the problem of Influence Maximization (IM). They use a greedy algorithm for selecting the most influential nodes as in previous works on IM, but the process of estimating the influence integrates a system of linear equations whose solutions are computed iteratively. They compute influence rank of nodes, and for each node added to the seed set they compute the influence estimation of the seed set, using Monte-Carlo simulations for their experiments, though they note that other techniques for influence estimation can be deployed. For the weighting of the graph’s edges the trivalency and the weighted cascade (WC) models are used.

## 5.2 Experimental Results

### 5.2.1 Twitter Data

We present Figures 3,4 to better understand the form of our Dataset in terms of number of users connected in the network and social relationships between users. In figure 3 we show the sets of users and their corresponding amount of friends. As can be observed from the figure, the majority of the users have friends that range from 1 to 100, while the amount of users with higher number of friends is small. In figure 4 the amount of tweets for the corresponding size of users is presented. This figure illustrates that the amount of tweets decreases as the amount of users increase and that only a few users have a great amount of tweets.

### 5.2.2 ESCAPE Evaluation

In this section we present the performance of our approach and we compare it with the state-of-the-art algorithm IRIE, which is the fastest algorithm in the literature that we know of and is able to perform influence maximization equally effectively to its competitors.

In the first set of experiments we present our approach when we set that 10% of the users are reachable (2371) out of the interested set (this corresponds to a set of 23710 nodes). In figure 5 we present the percentage of interested nodes that each approach is able to inform, relative to the users

informed if all reachable nodes were selected as seeds. ESCAPE manages to inform more users than IRIE at all times, with a percentage that ranges from 8% to 13%.

In the second set of experiments we have the same setting but we consider that 20% of the users is reachable (this corresponds to 4742 users). Figure 6 presents the percentage of interested nodes that were informed and we see that the percentage has decreased. That is because when sending to all 20%, more interested nodes would be informed, compare to 10% since the size of reachable users set is doubled. The above results are relative to the users that would be informed if all reachable nodes were used as seeds. However, the percentage of users that ESCAPE manages to inform is more than 9 units percent over IRIE at all times. As can be observed, as the number of seeds goes up the gap between ESCAPE and IRIE becomes slightly smaller. That is due to the fact that the intersection of seeds sets for both algorithms contains more nodes. We expect that they will converge later when all reachable nodes are added as seeds. We add 50 seeds per iteration, so that is why the angle is presented in figures 5 and 6. When no seeds are selected, no users are informed. IRIE requires about 4427 seconds for determining 500 seeds while ESCAPE requires only 385 seconds when reachable nodes are set to 20%, and 5158 seconds against 292 seconds respectively when reachable nodes are set to 10%. That makes ESCAPE more appropriate for emergency response. Due to the sparsity presented in the social graph, we notice a stabilized performance for both algorithms when more seeds are added.

## 6. RELATED WORK

Using social networks as dissemination tools has attracted interest in recent years in various application domains, including viral marketing campaigns and voting systems. In the majority of the applications the key point is influence maximization, *i.e.*, spreading the information to as many people as possible[13] or maximizing the likelihood that someone is being informed for a particular issue[10]. In all related works, the social network was represented as a weighted graph, with users as nodes and relationships between them as edges.

Several efforts have focused on inferring edge weights in so-

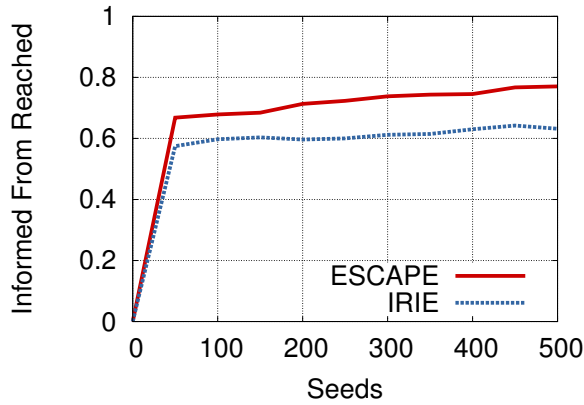


Figure 5: Informed From Reached - 10% Reachable

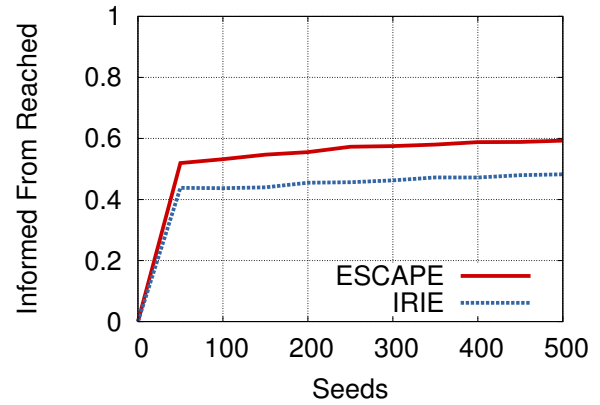


Figure 6: Informed From Reached - 20% Reachable

cial graphs. These take into account various information crawled by the social network related to users or the types of actions among the users. Recently, Facebook announced a metric called Edgerank, used to determine the items to populate at a users news feed. Edgerank considers the affinity of users, the actions made between users' profiles and time elapsed since the last interaction to determine the strength of the bond between different users and order news feeds according to the Edgerank metric[22]. Goyal *et al* [8] consider a number of models, including a Weighted Cascade (WC) model where the probability on an edge  $(v, u)$  is defined as  $1/in - degree(u)$ , a Trivalency model where probabilities are randomly and uniformly selected from the set  $\{0.1, 0.01, 0.001\}$ , and the model where probabilities learned from a training set using the Expectation Maximization (EM) based method suggested by Saito *et al* [20]. Unlike the Edgerank affinity, the number of users is not considered in this metric and different types of actions are not defined. Hangan *et al* in [10] suggest metrics for weighting the edges of a Social Graph so that the asymmetry presented in the network is captured. They also argue the fact that the shortest paths are eventually the strongest ones. They prove that utilizing edge weights may well improve global social search. Perhaps a more sophisticated metric that aims in predicting the strength of a relationship based on interaction and user similarity is established by Xiang *et al.* [25]. A latent variable model is considered in which the relationship strength forms the latent variable. An unsupervised model to estimate relationship strength from interaction activity and user similarity is developed. However, the model proposed requires data that may not be publicly available due to users privacy settings, thus, it is not applicable to all types of social networks.

Maximization of spread, given the weight of the edges already computed, is studied in [13]. The two models proposed are the Independent Cascade Model and the Linear Threshold model. The problem of influence maximization with the least effort (*i.e.*, using  $k$  or less nodes as starters), is stated as an NP-Hard problem, so provable approximation guarantees are obtained. In both models suggested by Kempe, nodes have two states, either active or inactive. When a node is active it means that the information has reached the node and the node is in position of forwarding it to its neighbors.

They explore the above described models when a set of  $k$  initial nodes is being activated (targeted). It is proved that a greedy hill-climbing algorithm for both models gives a good approximation as long as the influence function has certain properties which, as proved, is true. Influence probability and propagation is also a case of study in [3].

In [24] they propose variations for the independent cascade model so as to minimize computational cost. Paths whose probabilities is below a given threshold  $\theta$  are excluded, with  $\theta$  being tunable. Contrary to our approach, these efforts (1) assume that the influence graph is known and aim at maximizing the influence in the entire network rather than identifying and informing an appropriate subset of nodes most interested to the event, and (2) they focus on cases of viral marketing campaigns or voting systems, rather than emergency response situations.

Gomez-Rodriguez and Schölkopf [19] study the efficient influence maximization in time diffusing networks. They consider the influence maximization problem where information or influence can spread at different rates across different edges and analytically compute and efficiently optimize the influence avoiding the use of heuristics. The greedy algorithm in combination with the Lazy Evaluation, Localized Source Nodes and Limited Transmission Paths are proposed as speed-ups for the computations.

Kyungabaek *et al* [14] define two types of correlated to an event users. The ones located to the area of the event and those that have social ties with them. The notification system proposed for alerting users in case of an emergency event is aware of the geographies the message needs and social ties. Their system has a prior knowledge of Global Target Geography (GTG). Nodes are classified as physical nodes (PNodes) and trusted physical nodes (T-PNodes) that represent some sort of authority or public figure. After the occurrence of an event Possibly Affected Region (PAR) is defined as a sub-region of GTG and Possibly Damaged Region (PDR) as sub-region of PAR. They define two types of overlays. The Delivery Overlay which aims in reaching PNodes and the Information Overlay which is responsible for maintaining information about social entities. They customize the social diffusion process so that good initiators are selected.



## 7. CONCLUSIONS

In this paper, we have presented our ESCAPE system that investigates the relationships and interactions among the members of a social group, and develops a dissemination mechanism to maximize the information reach to a target set of users, when an emergency event occurs. As we illustrate in our experimental evaluation ESCAPE with its intelligent seed node selection process, manages to inform more interested users than the state-of-the-art technique IRIE, that aims in Influence Maximization.

## Acknowledgment

This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Programs: Aristeia-MMD Aristeia-Inception Investing in knowledge society through the European Social Fund, the FP7 INSIGHT project and the ERC IDEAS NGHCS project.

## 8. REFERENCES

- [1] F. Alvanaki, S. Michel, K. Ramamritham, and G. Weikum. See what’s enbogue: real-time emergent topic identification in social media. In *EDBT*, pages 336–347, Berlin, Germany, March 2012.
- [2] J. P. Bagrow, D. Wang, and A.-L. Barabasi. Collective response of human populations to large-scale emergencies. *CoRR*, abs/1106.0560, 2011.
- [3] F. Bonchi. Influence propagation in social networks: A data mining perspective. *IEEE Intelligent Informatics Bulletin*, 12(1):8–16, 2011.
- [4] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec. Whatsup decentralized instant news recommender. In *IPDPS*, Boston, MA, May 2013.
- [5] S. Das, Ö. Egecioglu, and A. El Abbadi. Anónimos: An lp-based approach for anonymizing weighted social network graphs. *IEEE Trans. Knowl. Data Eng.*, 24(4):590–604, 2012.
- [6] P. Eugster, P. Felber, and F. Le Fessant. The “art” of programming gossip-based systems. *SIGOPS Oper. Syst. Rev.*, 41(5):37–42, Oct. 2007.
- [7] R. Friedman, D. Gavidia, L. Rodrigues, A. C. Viana, and S. Voulgaris. Gossiping on manets: the beauty and the beast. *SIGOPS Oper. Syst. Rev.*, 41(5), Oct 2007.
- [8] A. Goyal, F. Bonchi, and L. V. Lakshmanan. A data-based approach to social influence maximization. volume 5, pages 73–84, Seattle, WA, August 2011.
- [9] M. Guy, P. Earle, C. Ostrum, K. Gruchalla, and S. Horvath. Integration and dissemination of citizen reported and seismically derived earthquake information via social network technologies. In *IDA*, pages 42–53, Tucson, AZ, 2010.
- [10] S. Hangal, D. MacLean, M. S. Lam, and J. Heer. All friends are not equal: Using weights in social graphs to improve search. In *SNA-KDD*, Washington, DC, July 2010.
- [11] M. Iqbal, X. Wang, D. Wertheim, and X. Zhou. Swanmesh: A multicast enabled dual-radio wireless mesh network for emergency and disaster recovery services. *JCM*, 4(5):298–306, 2009.
- [12] K. Jung, W. Heo, and W. Chen. Irie: Scalable and robust influence maximization in social networks. In *ICDM*, pages 918–923, Brussels, Belgium, December 2012.
- [13] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, pages 137–146, Washington, DC, USA, August 2003.
- [14] K. Kim, Y. Zhao, and N. Venkatasubramanian. Gsford: Towards a reliable geo-social notification system. In *SRDS*, pages 267–272, Irvine, CA, October 2012.
- [15] B. Krishnamurthy, P. Gill, and M. Arlitt. A few chirps about twitter. In *WOSN*, pages 19–24, Seattle, WA, USA, August 2008.
- [16] A. M. Lesperance, M. A. Godinez, and J. R. Olson. *Social networking for emergency management and public safety*. Pacific Northwest National Laboratory Richland, WA, 2010.
- [17] B. Liu, G. Cong, D. Xu, and Y. Zeng. Time constrained influence maximization in social networks. In *ICDM*, pages 439–448, Brussels, Belgium, December 2012.
- [18] J. J. Ordille, P. Tendick, and Q. Yang. Publish-subscribe services for urgent and emergency response. In *COMSWARE*, pages 8:1–8:10, Dublin, Ireland, June 2009.
- [19] M. G. Rodriguez and B. Schölkopf. Influence maximization in continuous time diffusion networks. In *ICML*, Edinburgh, Scotland, June 2012.
- [20] K. Saito, R. Nakano, and M. Kimura. Prediction of information diffusion probabilities for independent cascade model. In *KES*, pages 67–75, Zagreb, Croatia, Sep 2008.
- [21] J. Singh and J. Bacon. Event-based data control in healthcare. In *Middleware*, pages 84–86, Leuven, Belgium, December 2008.
- [22] D. Taylor. Everything you need to know about facebook’s edgerank. *The Next Web*, 9, May 2011.
- [23] G. Valkanas and D. Gunopulos. Location extraction from social networks with commodity software and online data. In *ICDMW*, Brussels, Belgium, December 2012.
- [24] C. Wang, W. Chen, and Y. Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25(3):545–576, 2012.
- [25] R. Xiang, J. Neville, and M. Rogati. Modeling relationship strength in online social networks. In *WWW*, pages 981–990, Raleigh, NC, USA, April 2010.

# Crowdsourcing turning restrictions for OpenStreetMap

Alexandros Efentakis  
Research Center “Athena”  
Artemidos 6, Marousi 15125, Greece  
efentakis@imis.athena-innovation.gr

Nikos Grivas  
Research Center “Athena”  
Artemidos 6, Marousi 15125, Greece  
grivas@imis.athena-innovation.gr

Sotiris Brakatsoulas  
Research Center “Athena”  
Artemidos 6, Marousi 15125, Greece  
mprakats@ceid.upatras.gr

Dieter Pfoser<sup>\*</sup>  
Department of Geography and  
GeoInformation Science  
George Mason University  
4400 University Drive, MS 6C3  
Fairfax VA 22030-4444  
dpfoser@gmu.edu

## ABSTRACT

The abundance of GPS tracking data due to the emergence and popularity of smartphones has fuelled significant research around GPS trajectories and map-matching algorithms. Unfortunately, none of this previous research addresses the issue of identifying turning restrictions in the underlying road network graph. Our latest research endeavour remedies this, by proposing a novel, straightforward and effective way to infer turning restrictions for OpenStreetMap data, by utilizing historic map-matching results from an existing fleet management service. Our experimental evaluation based on the results acquired for three European cities within an one-year period, proves the robustness and credibility of our method.

## Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: Information filtering;  
H.2.8 [Database Applications]: Spatial databases and GIS

## General Terms

Design

## Keywords

Crowdsourcing, Turning Restrictions, Map-matching, OpenStreetMap

## 1. INTRODUCTION

Our latest research efforts of [22] aimed towards combining state-of-the-art research about road networks, Floating Car Data, map-matching, historic speed profile computation, live-traffic assessment and time-dependent shortest-path computation to provide an efficient, yet economical fleet management solution. This process

<sup>\*</sup>On leave from Research Center “Athena”, Greece.

was documented in [5] and its result, our fully functional SimpleFleet fleet management system and its accompanying demo [7] now cover the urban regions of three European metropolitan cities namely: Athens (Greece), Berlin (Germany) and Vienna (Austria). Creating the actual service required several intermediate steps such as: Creating road network graphs from OpenStreetMaps data, collecting a large amount of Floating Car Data (FCD) from fleet vehicles, applying state-of-the-art map-matching algorithms on this data for aligning the GPS traces to the road network graph and consequently producing high-quality historic speed profiles along with frequently updated live-traffic assessment. This combination of live-traffic information and speed profiles was subsequently used to provide up-to-date live-traffic shortest-path and isochrone computation (refreshed every 5 minutes). In addition, our recent work of [6] has already combined the live-traffic isochrone functionality of this system with demographic / business data to showcase the impact of traffic fluctuations in a geomarketing context. There, one can see in a quantitative way the considerable importance of traffic and how it should affect geomarketing decisions.

It is obvious that the success of every fleet management solution depends highly on the quality of the road network graph. Although OpenStreetMap (OSM) is a crowdsourced, high-quality, frequently updated road network dataset, an entire year of running the SimpleFleet service for its three urban regions has revealed a inherent limitation of the OSM dataset: Its limited information for turning restrictions, i.e., a transition from one network edge to another (via an intersection node) that is prohibited due to local traffic rules. It is not that OSM data does not support turning restrictions: an additional relation tag (Relation:restriction [19]) is defined for describing such restrictions. The problem is that only a small number of users contribute to this information. While OSM includes more than 2.1 billion Nodes, Ways and Relations [17], less than 230,000 relations actually represent turning restrictions [19]. This is even more obvious, when we look at our individual test cases: For the city of Athens and its road network of 277K nodes, only 214 turning restrictions have been recorded by OSM users. This lack of data is to be expected: there are no public datasets for traffic signs easily found (if any), satellite imagery cannot reveal this information and adding turning restrictions even for a single road is extremely time-consuming. Keep in mind that turning restrictions do not include one-way streets. Such streets are easily modeled in every directed graph representation, are easily recognized by users and OSM data has extensive coverage for them.

Turning restrictions on road networks are especially important for any routing / isochrone service. While a lot of scientific literature has focused on time-dependency on road networks (due to the fluctuation of traffic) and consequently the implementation of efficient shortest-path algorithms that address this issue, there is a limited number of works that deal with turning restrictions. This is mostly due to the fact that “*no publicly-available realistic turn data exist*”[4]. Note that turning restrictions have a more dramatic impact on shortest paths provided by mapping services than traffic: While ignoring traffic returns a valid, yet suboptimal route to the user, ignoring turning restrictions returns erroneous paths that may lead to accidents. As a result, providing an efficient method for automatically identifying turning restrictions is extremely important.

When searching existing scientific literature for solving this issue, we found a significant body of work based on Floating Car Data (FCD) in various areas (see [29] for a partial overview on GPS related research). The only previous works relevant to solving (or even acknowledging) our actual problem also use FCD for calculating turn delays [1, 13, 23, 27, 28]. It was really a surprise, when we did not find any literature devoted to the results of the map-matching (MM) algorithms, as though those results may strictly be used for travel-time estimation. In this sense, we beg to differ, because the main focus of this work is to automatically identify / infer turning restrictions in the OSM dataset by utilizing historic map-matching results, i.e., we crowdsource the identification of turning restrictions to local vehicle drivers by mining the map-matching results produced by them, when traversing the road network graph. This is also the true novelty of our contribution: instead of using the GPS trajectories directly, we use the map-matching results derived from them. Our approach makes sense: In comparison to raw GPS traces, map-matching results are: a) more condensed, since instead of random locations in the plane we have edge sequences and b) less error-prone (if an efficient map-matching algorithm is used) since they are interpolated with the actual road network. Therefore it is logical to utilize those historic results to extrapolate this additional meaningful information, instead of using raw FCD like most previous works. To the best of our knowledge, we are the first to utilize map-matching results for such a task. Although our method uses OSM data, it may also be used for any road network dataset, in cases where the road network evolves faster than typical map updates. In those cases, identifying added turn-restrictions, as soon as they occur, is extremely important.

The outline of this work is as follows. Section 2 describes previous research in relation to our work. Section 3 describes our scientific contribution towards identifying turning restrictions in the OSM dataset by utilizing historic map-matching results. Section 4 summarizes the results of our approach. Finally, Section 5 gives conclusions and directions for future work.

## 2. RELATED WORK

Recently, real-time Floating Car Data (FCD) collected by operating vehicles equipped with GPS-enabled devices has become the mainstream in traffic study because of its cost-effectiveness, flexibility and being the “*the only significant traffic data source with the prospect of global coverage in the future*”[10]. Typically a GPS trajectory describing a vehicle movement, consists of a sequence of measurements with latitude, longitude and timestamp information. However, this data is inherently imprecise “*due to measurement errors caused by the limited GPS accuracy and the sampling error caused by the sampling rate*” [20]. Therefore the observed GPS positions often need to be aligned with the road network graph. This process is called map-matching. As a result, a map-matching (MM) algorithm accepts as input a vehicle’s GPS trajectory and outputs a

path / ordered sequence of road network graph edges that this vehicle has traversed, along with travel time information, i.e., how long did it take for the specific vehicle to traverse the calculated path. In our SimpleFleet service [5] we used two different MM algorithms: the Fréchet-distance-based curve matching algorithm of [2, 25] and the [11] implementation of the ST-matching algorithm [14]. However, both implementations were significantly enhanced to handle live incoming FCD streams.

Despite their inherent imprecision and the usually low sampling rate of available datasets, latest years saw an explosion of research around GPS trajectories ([29] presents a partial overview of GPS related research). Nevertheless, so far, only a limited portion of this research focused on road network intersections. This is a major oversight, since intersections are important components of urban road networks and contribute much to the total travel time cost [16, 24]. [16] concludes that intersection delays i.e., the turn cost associated with the continuation of travel between edges via an intersection node [26] contribute to 17-35% of the total travel time, according to a conducted survey in the Copenhagen urban area.

The few research works around road network intersections that actually exist, focused on estimating intersection delays based on the available Floating Car Data. Some researchers have utilized the historical mean method to calculate the intersection delays ([23], [28]), while other authors employ piecewise linear interpolation ([1], [27]). Additional works employed the principal curves method [9] to overcome data sparseness of Floating Car Data and calculate turn delays tables for the region of Beijing [13].

Although turn costs / intersection delays are a generalization of turning restrictions (i.e., a turning restriction is a turn with delay set to  $\infty$ ) previous works are fundamentally different from our approach in several levels. First of all, for previous approaches to calculate turn cost for a specific turn, many vehicles need to actually use it. On the contrary, we identify turning restrictions by focusing on turns with no available data. Second, previous approaches use GPS trajectories; we use map-matching results. Third, since they are based on data mining techniques they may only verify results by dividing the original GPS dataset into a training and a test set. We use an independent mapping service to verify our findings. Lastly, most publicly available GPS datasets are either simulated [12], focused on a specific city [3, 15] or for limited time periods (a day, week or month for [12, 15, 3] respectively). Contrarily, our conclusions are based on three different European cities and fleets of 2,000 – 5,000 vehicles per city covering a full 12-month period. Since our results are almost identical for each city (see Sec. 4) it is obvious that our method is both realistic and robust.

In the next section we will describe the basic methodology with all the practical details of our implementation.

## 3. CROWDSOURCING TURNING RESTRICTIONS

In this section we are going to present basic information about the OpenStreetMap dataset and the methodology we used to infer information about turning restrictions from historic map-matching results, as well as the verification process used.

### 3.1 Preliminaries

In the discussion that follows, a road network is represented by a directed weighted graph  $G(V, E, l)$ , where  $V$  is a finite set of vertices / nodes,  $E \subseteq V \times V$  are the edges of the graph and  $l$  is a positive weight function  $E \rightarrow R^+$ . Typically, the weight  $l$  represents the travel time required to traverse the edge. In other cases,  $l$  may refer to the length of the edge in meters (for travel distances metric).



**Figure 1: Prohibitory traffic signs for turning restrictions**

The degree of a vertex  $u$ , denoted as  $deg(u)$ , is the number of edges incident to the vertex. *Intersection nodes* are the road network vertices with node degree larger than two, i.e.,  $I = \{v_i \in V, deg(v_i) > 2\}$ . A *turning restriction* is an ordered sequence of two or more network edges connected via intersection nodes that is prohibited due to local traffic rules. Such turning restrictions are represented in the road network as traffic signs (see Fig. 1). For the remainder of the paper, we only deal with those edge sequences that consist of a single ordered pair of two edges connected via a single intersection node, since those represent the majority of turning restrictions on typical road networks. Note that turning restrictions do not refer to one-way streets, because a) even a single edge may be marked as unidirectional and b) turning restrictions may refer to roads that are bidirectional but it is only their sequence that is prohibited. In addition, unidirectional streets are easy to model in every directed graph representation, whereas turning restrictions is a distinguishing trait of road networks that differentiates them from most other real-world networks.

A GPS trajectory describing a vehicle movement, consists of a sequence of measurements with latitude, longitude and timestamp information for the same vehicle ID. *Map-matching* is the process of aligning such a trajectory against the underlying road network graph. Consequently, a map matching algorithm accepts as input a single vehicle’s trajectory and outputs a path / ordered sequence of road network graph edges that this vehicle has traversed, along with travel time information, i.e., how long did it take for the specific vehicle to traverse the calculated path. For the remainder of the paper, when we are talking about map-matching results we only refer to the calculated vehicle’s path and not the associated travel time information.

## 3.2 OpenStreetMap and turn restrictions

OpenStreetMap [18] is a free, editable map of the entire world. Unlike proprietary datasets, the OpenStreetMap license allows free access to the full map dataset. This massive amount of data may be downloaded in full but is also available in other useful formats such as mapping, geocoding or other web services. Users participate in the OpenStreetMap (OSM) community by providing feedback and editing the map. Although OpenStreetMap contains an appropriate relation tag (Relation:restriction [19]) for describing turning

**Table 1: Turning restrictions added in OSM per year for the cities covered by our service**

city	2009	2010	2011	2012	2013	Total
Athens	-	11	1	75	127	214
Berlin	8	26	101	386	147	668
Vienna	33	36	99	307	324	799

**Table 2: OSM road networks of the three cities covered by our service**

city	# vertices	# edges	# intersection vertices		# intersection vertices for roads $\leq 10$	
			total	%	total	%
Athens	277,719	329,444	100,422	26%	34,921	13%
Berlin	89,598	103,486	51,935	58%	21,119	24%
Vienna	100,579	112,478	44,874	45%	16,104	16%

restrictions, only a small number of OSM users contribute to this information, due to its inherent difficulty. This statement was easy to confirm for the three European cities (Athens, Berlin, Vienna) covered by our service. Results retrieved in September 2013 are shown in Table 1.

Table 1 shows that available data for turning restrictions is particularly low, especially in comparison to the sizes of the OSM road networks of the three cities covered by our service, as shown in Table 2. We got similar results (or even worse) for other European cities, especially for countries with less detailed maps (e.g., Albania, Montenegro). Given the above issue, we decided to take advantage of the large amount of historic map-matching results created through our fleet management service and infer / identify turning restrictions in the OSM road network. Our method is straightforward and efficient: We aim to discover turns that, although they are allowed in the original dataset, in practice they are rarely, if ever, used by the vehicle drivers. Such turns that exhibit such low usage frequency have a very high probability to be actually prohibited.

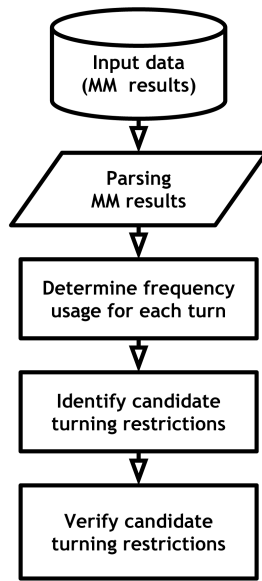
## 3.3 Methodology

The basic methodology for inferring / identifying OSM turning restrictions by using historic map-matching results created by our fleet management service may be described by the following simplified diagram of Fig. 2. The independent stages of this process will be thoroughly discussed in the following sections.

### 3.3.1 Input data

In our SimpleFleet service [5], GPS traces of fleet vehicles for the three European cities, arrive in a streaming fashion. Specifically, for each of those urban areas, we are dealing with 2,000-5,000 fleet vehicles producing a data point (GPS position sample) every 60 -180s. GPS trajectories for each vehicle are subsequently map-matched, in order to align those GPS traces to the underlying OSM road network graph. The result of this process is an ordered sequence of road edges that a vehicle has traversed. The traffic data-store of the service, including the OSM road network graphs, Floating Car Data and map-matching results is implemented by PostGIS enabled PostgreSQL databases (one database instance per city) for permanent storage.

As a vehicle moves in the road network, it traverses roads of varying importance to traffic (refer to Table 3 for the distribution of the OSM road networks per their respective category for the three SimpleFleet cities). However, not all these roads are important to



**Figure 2: Methodology for identifying OSM turning restrictions by using historic map-matching results**

**Table 3: Road categories for the OSM road networks**

CategoryID	Road category	Athens	Berlin	Vienna
1	motorway	4287	1420	2410
2	motorway link	3747	2012	4386
3	trunk	1343	111	171
4	trunk link	567	0	227
5	primary	16210	5203	8913
6	primary link	1257	347	422
7	secondary	42881	21250	12894
8	secondary link	0	45	0
9	tertiary	58722	9678	11576
10	tertiary link	0	6	0
11	unclassified	13484	2792	3060
12	road	395	28	0
13	residential	186459	58338	67482
14	living street	92	2256	937

traffic, so it made sense to reduce the bulk of data stored. Towards this goal, a separate process eliminates map-matched edges that belong to edges of road category greater than 10 (i.e., OSM categories for unclassified, road, residential and living street - see Table 3). Depending on the time period and the traffic patterns in each city, about 12-15% of the map-matched records are subsequently dropped after the map-matching process.

Since map-matched records are primarily used to offer real-time information for the current traffic situation, older data is periodically removed from the respective PostgreSQL datastores (every 5 minutes) and archived into csv files for offline use. At the end of each day, a batch process compresses those csv files created during the day. A copy of this compressed file is then sent to a backup server for permanent storage. Table 4 indicates the typical size of compressed archives produced per day and month for each city.

After one year of running the service (from Oct 2012 to end of September 2013), we have accumulated several Gb of compressed historic map-matching results for each of the cities covered by our service. The challenge is how to utilize this significant wealth of

**Table 4: Typical size of compressed MM results archives**

Size	Athens	Berlin	Vienna
per day	22.3 MB	224 MB	76.3 MB
per month	0.67 GB	6.74 GB	2.29 GB

**Table 5: Total counted instances for all examined turns between Oct 2012 and September 2013**

city	# intersection vertices for roads $\leq 10$	#examined turns	# total instances	# instances per inters. vertex for roads $\leq 10$
Athens	34,921	75,552	144,451,729	4,137
Berlin	22,119	44,636	2,054,969,090	97,304
Vienna	16,104	36,484	610,902,632	37,935

data to infer turning restrictions for the respective OSM road networks. This process will be described in the following sections.

### 3.3.2 Parsing map-matching results and optimizations

The basic focus of our work is to identify those turns (i.e., ordered pair of edges connected via an intersection node) that exhibit unusually low frequency of usage by vehicles. The frequency of usage will be determined by parsing the compressed archives of the historic map-matching results produced for the three cities during the operational period of our service (Oct 2012 to end of September 2013). Since, the respective OSM road networks comprise of hundreds of thousand of nodes and edges (see Table 2) we need to somehow limit the possible turns that need to be examined.

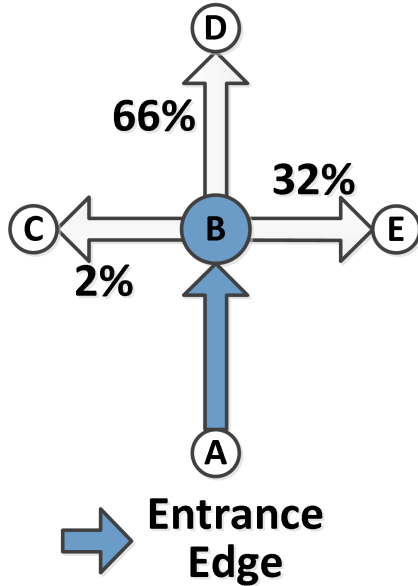
The first optimization is to identify those pairs of consecutive edges that connect at intersection vertices. There is no need to accumulate information for vertices of degree 2 (with just one incoming and one outgoing edge) or lower, since in those vertices the driver has no choice but to continue in one direction. The second optimization had to do with the available input data. Since map-matching results only include larger roads that correspond to OSM categories  $\leq 10$  (see Section 3.3.1), we are only interested in those intersection vertices connected to such roads. This way we miss some intersection vertices (strictly connected to smaller categories roads) but this is a necessary compromise to minimize our scope. In addition, intersection vertices that connect to major roads are more likely to be used by many vehicle drivers and as a result they significantly influence traffic behaviour. Table 2 shows that the number of intersection vertices connecting major roads are less than 25% of total vertices for all cities covered by our service.

As a result of those two optimizations, the number of unique turns / pairs of consecutive edges we need to examine is significantly smaller than the available road network edges, which is a considerable improvement (see Table 5). Since the OSM road networks of each city are stored in the respective PostgreSQL datastores, determining intersection vertices of interest and their corresponding turns is easily accomplished with plain SQL commands.

After determining the unique turns for examination, we implemented a custom Java app that parses the compressed archives of historic map-matching results in our disposal (see Section 3.3.1), counts the instances encountered for each turn and stores results in the respective PostgreSQL datastores. The total counted instances for all examined turns during our one-year testing period (starting Oct 2012) are shown in Table 5. Results also show, that on average for every intersection vertex connected to major roads (i.e., their road category  $\leq 10$ ), we have 4,137 (Athens) - 97,304 (Berlin) counted instances of turns, which means that we have a sufficiently large number of measurements per intersection vertex.

### 3.3.3 Identifying candidate turning restrictions

After enumerating instances for every unique turn we needed to examine, we must discover which of those turns are rarely used. Since both turns and results of the enumerating process are stored in the respective datastores, it is easy to group results / turns by *entrance* edge and direction (for bidirectional edges). Each such group contains all possible turns a vehicle may follow after traversing a specific entrance edge (and a specific direction). Subsequently, each turn belongs to only a single group of turns. Since we know the number of instances encountered for each one of the turns belonging to the same group, it is easy to calculate the percentage of usage for each one. An example group for a specific entrance edge is shown in Fig. 3.



**Figure 3: A simple example of grouping turns per entrance edge (A, B) at an intersection vertex (B) for calculating usage percentage per turn**

As we notice in the example group of Fig. 3, most drivers continue straight when they traverse the entrance edge (A, B) that leads to the intersection vertex B. Some others prefer to turn right. But a very small percentage of them (2%) turn left. This is a very strong indication that this low percent actually represents erroneous map-matching results (even the most efficient MM algorithm has a small error rate) and indeed this particular left-turn is prohibited, even if OpenStreetMap lacks this information.

Next, we made the rather logical assumption that turns with lower frequency percentage than an implicit 5% threshold are probably prohibited. Of course this threshold is arbitrary but as results will show, it is pretty accurate as well. Table 6 shows the number of the candidate turning restrictions we have discovered for each city for both 5% and 2.5% thresholds.

However, estimating candidate turning restrictions is not enough. For each such turn, we need to additionally calculate its direction, to conclude if it is a straight, right, left or U-turn. The direction calculation is very easy, since we have already stored the inclination of each edge in the respective datastore, since this information was needed for the isochrone functionality of our service. Table 7 shows the percentages of the turns direction categorization for the candidate prohibited turns we have extracted. As expected, most of them (particularly in Berlin and Vienna) represent left-turns.

**Table 6: Number of candidate turning restrictions discovered for 5% and 2.5% thresholds**

city	# turns	# turning restrictions		turning restrictions (%)	
		5%	2.5%	5%	2.5%
Athens	75,552	5,287	3,596	7.00%	4.76%
Berlin	44,636	2,653	1,582	5.94%	3.54%
Vienna	36,484	1,739	1,261	4.77%	3.46%

**Table 7: Categorization of candidate turning restrictions per direction for 5% threshold**

city	# turning restrictions	straight	left	right	U-turn
Athens	5,287	6.5%	45.4%	41.6%	6.5%
Berlin	2,653	1.6%	64.6%	18.6%	15.2%
Vienna	1,739	10.5%	44.8%	30.0%	14.7%

Still, after determining the candidate prohibited turns for each city covered by our service, we still need to find additional means to verify the validity of our claims. This process is discussed in the following section.

### 3.4 Verifying results

There are two basic ways to validate the discovered candidate turning restrictions. Firstly, we can visualize each one of those candidate restrictions. Secondly, we can use an external mapping service and cross-check if we get similar results to ours. We used both ways: Results are presented in the following sections.

#### 3.4.1 Visual Inspection

In order to confirm our results, we need to visualize the candidate restricted turns. Each such turn may be represented with the appropriate traffic sign (depending on the direction of the turn according to Table 7) located in the corresponding intersection vertex coordinates. For that purpose, we used QGIS [21], a popular, free and open source GIS application that runs in all major operation systems. We used a Google Maps Layer in QGIS as the background map layer, in order to compare results to an external mapping service. Figure 4 shows some typical examples of the results of our visualization process for some of the candidate turning restrictions:

- Figure 4(a) depicts an intersection familiar to most local drivers in the center of Athens. This type of restrictions were easily verified by our personal experience and they effectively demonstrate how important and critical turning restrictions are discovered through our method.
- Figure 4(b) shows a case of a prohibited U-turn in the Berlin area. There, many disallowed U-turns are missing from the OpenStreetMap dataset.
- Figure 4(c) shows that the Google Maps layer visually confirms the discovered turning restriction.

Conclusively, visual inspection of our findings shows that OpenStreetMap data (despite its high quality) is still missing some vital information, especially as turning restrictions are concerned. By taking advantage of historic map-matching results, we have tracked and visualized many such problematic cases. As a result, our methodology may be used to further enrich the collection of turn-restrictions available in OpenStreetMap. Yet, to further verify and quantify results, an additional high quality mapping service could be used for cross-checking the validity of our findings. This process will be described in the following section.



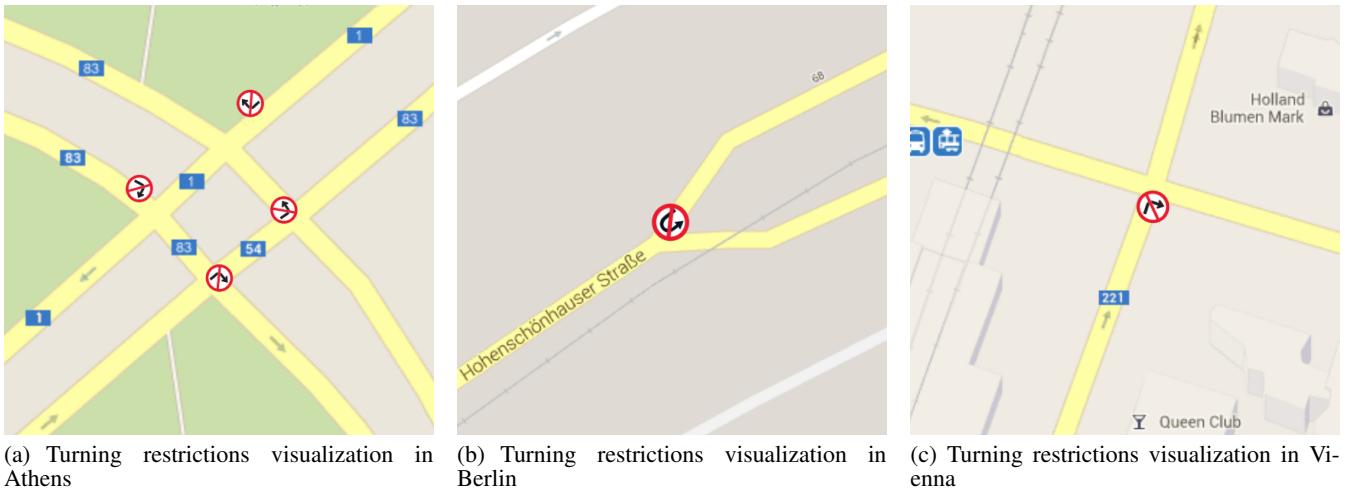


Figure 4: Visualizing turning restriction with QGIS

### 3.4.2 Sourcing an external mapping service

Although visual inspection is a convincing, qualitative way to validate results, it would be best if we could further verify and quantify our findings through an automatic process. One solution to this problem is to use the Google Directions API [8]. Although Google Maps is not guaranteed to be perfect, yet, it is a global, popular, commercial, alternative solution to the crowdsourced user-generated data of OpenStreetMap.

The Google Directions API is a service that calculates directions between locations using HTTP requests. Users may search for directions for several transportation modes, include transit, driving, walking or cycling. Directions may specify origins, destinations and waypoints either as text strings or as latitude/longitude coordinates. The Directions API can return multi-part directions using a series of waypoints [8].

The Google Directions API allows only 2,500 directions requests per 24 hour period from a single IP address (for free users). This is the reason, why it was important to first identify (a rather limited number) of candidate turning restrictions, so that the required requests to the API could finish in a few days. In any such HTTP request to the API, certain parameters are required while others are optional. As is standard in URLs, all parameters are separated using the ampersand (&) character. The most important required parameters (relative to our problem) are:

- Origin - The address or textual latitude/longitude value FROM which we wish to calculate directions.
- Destination - The address or textual latitude/longitude value TO which we wish to calculate directions.

Two additional, optional parameters useful to our purpose are:

- Mode (defaults to driving) - Specifies the mode of transport to use when calculating directions. The value can be “driving”, “walking”, “bicycling” or “transit”.
- Waypoints - Specifies an array of waypoints. Waypoints alter a route by routing it through the specified location(s). A waypoint is specified as either a latitude/longitude coordinate or as an address which will be geocoded.

Since our requests concern driving directions, there is no need to specify the “mode” parameter as it defaults to “driving”. As for waypoints, we only need one waypoint per request. This waypoint

```
http://maps.googleapis.com/maps/
api/directions/json?origin={A_
coordinates}&destination={C_
coordinates}&waypoints=via:{B_
coordinates}&sensor=false
```

Figure 5: A sample Google Directions API request

should not be a stopover but serves just to influence the route. This may be done by prefixing the waypoint with the prefix “via:” (in the respective API call). This way, a single-part route is returned.

Given the above and with reference to Fig. 3, in which the  $Turn(A \rightarrow C \text{ via } B)$  has a low frequency usage, an HTTP request to verify this candidate turning restriction would be similar to Fig. 5. This request returns a JSON object with the proposed route by the API. The process for verifying the turning restriction is described in Algorithm 3.1, which compares the distance (in meters) calculated by the Google Directions API with the sum of lengths of edges  $(A, B)$  and  $(B, C)$ . If the Google distance is significantly greater (over 10%) than OSM’s distance, then we may safely assume that indeed there is a turning restriction and the Google Directions API has to follow a much longer route than simply  $(A, B) \rightarrow (B, C)$ .

```
Algorithm 3.1: VERIFYTURN( $Turn(A \rightarrow C \text{ via } B)$ )
 $GooglePath \leftarrow DirectionsAPICall(A \rightarrow C \text{ via } B)$ 
if  $dist(GooglePath) \gg dist(A \rightarrow B) + dist(B \rightarrow C)$ 
then  $\{Turn(A \rightarrow C \text{ via } B) \text{ is verified}$ 
```

In order to access the Google Directions API, we implemented a Java command-line application that retrieves turns below a threshold frequency usage (5% in our case) from the datastore, constructs an appropriate request string similar to Fig. 5 for each turn and retrieves the distance of the route returned by the API. To avoid overloading Google’s servers and getting rejected requests, we have enforced a 500 ms gap between requests. API distance results are also stored in the respective PostgreSQL datastore for easy accessing.

An obvious problem to this approach for verifying results, is the usage limits of the Google Directions API. Although we are dealing with road networks with hundreds of thousands of nodes, edges and



**Table 8: Number of verified restrictions for 5% and 2.5% implicit threshold**

city	candidate turning restrictions		# verified		verified (%)	
	5%	2.5%	5%	2.5%	5%	2.5%
	Athens	5,287	3,596	3,517	2,471	67%
Berlin	2,653	1,582	1,510	1,016	57%	64%
Vienna	1,739	1,261	1,172	880	67%	70%

possible turns, through our optimizations (see Section 3.3.2) and by restricting the usage of the API to strictly confirm the candidate prohibited turns found by our proposed method, we only need to check a few thousands turns. Even by not bypassing Google API’s limits (by using different IP addresses) this process only takes 1-3 days per city (e.g., for Vienna it requires only a few hours). As a result, the API usage limits easily suffice for confirming our results. These results are presented in the following section.

## 4. RESULTS

This section summarizes the results produced by the Google Directions API verification process for all candidate turning restrictions in comparison to the original OpenStreetMap datasets.

### 4.1 Verified turning restrictions

The method used for comparing results of Google Directions API and the OSM distances for each turn was thoroughly explained in Section 3.4.2. In Table 8, we present the number of restrictions verified for both 5% and 2.5% implicit usage threshold, as well as their respective percentages in comparison to the total candidate restrictions. Keep in mind that usually the paths returned by the Google Directions API are significantly larger (85-90% of the verified restrictions give at least two-times larger paths) than the sum of lengths (A, B) and (B, C), which further proves the validity of the verification method used.

As we notice, the majority of the candidate restrictions are successfully verified by the Google Directions API. In fact, in Athens and Vienna more than 67% of the extracted turning restrictions are verified. In Berlin, the verified restrictions are about 57% for the 5% threshold and 64% for the 2.5% threshold. Another useful remark is that moving from the 5% to the 2.5% threshold, the verified restrictions’ percentage is slightly increased but, in fact, we are missing a significant number of restrictions (compare columns “# verified” for 5% and 2.5%). This means, that there is a respectable amount of existing (and verified) restrictions even in the turn usage interval between 2.5% and 5%.

Finally, Table 9, compares total turns, examined turns, candidate and verified turning restrictions in comparison to the turning restrictions existing in the original OpenStreetMap datasets for the three respective cities. Results are quite impressive: Instead of examining hundreds of thousands of turns, by focusing on intersection nodes connecting major roads and utilizing historic map-matching results, we discovered only a few thousand candidate turning restrictions in need of verification. Next, by using the Google Directions API most of the candidate turning restrictions were successfully verified. But the most impressive fact of all, is that the number of verified turning restrictions is significantly larger than the restrictions existing in the original datasets. Especially in Athens, the number of verified turning restrictions is 16 times larger than those existing in the original OSM dataset. Even, in Vienna and Berlin the number of the verified prohibited turns is still 1.7 - 2.2 times larger than those existing in the original data. Our

**Table 9: Total turning restrictions results for 5% implicit threshold in comparison to existing OSM’s restrictions**

city	total turns	examined turns	candidate turning restrictions	verified turning restrictions	OSM turning restrictions
Athens	900,397	75,552	5,287	<b>3,517</b>	214
Berlin	252,271	44,636	2,653	<b>1,510</b>	668
Vienna	256,185	36,484	1,739	<b>1,172</b>	799

results lead us to assume that in cities of European countries with less detailed maps (e.g., Albania, Montenegro) the situation will be similar to Athens or even worse.

### 4.2 False positives?

Another pending question is what can we really infer for those turning restrictions that were not verified by the Google Directions API. Most of the times, for those turns, the distance returned by the Google Directions API is quite similar to the sum of lengths (A, B) and (B, C). Still, there is also a non-negligible number of routes (5% for Athens, 2% for Berlin and 8% for Vienna of those unverified restrictions) where the distance returned by the API is less than 80% of the sum of lengths of edges (A, B) and (B, C). When we searched for those strange cases, most of the times there were serious inconsistencies between the two maps. In that case, if the turn is actually allowed or not is very debatable.

Still, even if we assume that all unverified turning restrictions are indeed allowed (i.e., our method produces false-positives) there is one fact we cannot ignore: *A very small percent of drivers actually use them.* In that case, a perfect shortest-path solution *would still have to penalize (by increasing the respective turn cost) such “unappealing” turns.* In this sense, even unverified turning restrictions returned by our method are still useful in revealing typical drivers’ patterns and behaviors.

## 5. CONCLUSION AND FUTURE WORK

In this work we have proposed a new and efficient, semi-automatic way to infer / identify turning restrictions for OpenStreetMap data by utilizing historic map-matching results from an existing fleet management service, covering three major European cities, spanning a period of twelve months. Our method has proved solid: 57-67% of the turning restrictions we have extracted may be successfully verified. However, the most important result is that we have identified and verified 2-16 times more turning restrictions than those existing in the original datasets. This impressive feat proves the validity and credibility of our method.

To the best of our knowledge, we are the first to utilize historic map-matching results for such a task. This is after all, the main novelty of our work, since the few existing works that deal with the similar subject of intersection delays base their research on raw GPS trajectories. In addition, most previous works used either simulated data or data covering smaller time periods (up to a month) and were focused on a particular area. Our results are based on three European cities, originate from three medium / large fleets of 2,000-5,000 vehicles per city and cover an entire year of operation. Results for the three areas were almost identical, which further proves the robustness of our method. Moreover, by comparing our results with an external mapping service (the Google Directions API) we have shown the correctness of our approach. On a quite similar note, we also experimented with two fundamentally different map-matching algorithms and our results showed that our method produces similar results regardless of the map-matching algorithm used.

We can give the following directions for future work. Since the proposed method is able to identify and confirm turning restrictions in the OSM data we can expand it to automatically contribute those confirmed restrictions back to the OpenStreetMap project. This way, the product of our work could be shared by the related mapping community. Additionally, our results could be proven extremely useful to further improving the quality of existing map-matching algorithms. Many of those algorithms use partial shortest-path calculations to align the raw GPS traces to the road network graph. Up until now, those SP computations do not take turning restrictions into account. Since, our approach identifies such restrictions, those newly found constraints could be integrated back in the map-matching algorithms to further improve their results. That way a self-improving, evolutionary map-matching algorithm might be possible after all.

## Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme “SimpleFleet” (<http://www.simplefleet.eu>, grant agreement No. FP7-ICT-2011-SME-DCL-296423).

The authors would additionally like to thank Kostas Patroumpas for his work on the map-matching algorithms and his useful insight.

## 6. REFERENCES

- [1] J. Ban, R. Herring, P. Hao, and A. M. Bayen. Delay pattern estimation for signalized intersections using sampled travel times. *Transportation Research Record*, pages 109–119, 2009.
- [2] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proc. 31st VLDB Conference*, pages 853–864, 2005.
- [3] Complex Engineered Systems Lab. Taxi Trajectory Open Dataset [Online]. <http://sensor.ee.tsinghua.edu.cn/datasets.php>, 2010.
- [4] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Proceedings of the 10th international conference on Experimental algorithms*, SEA’11, pages 376–387, Berlin, Heidelberg, 2011. Springer-Verlag.
- [5] A. Efentakis, S. Brakatsoulas, N. Grivas, G. Lamprianidis, K. Patroumpas, and D. Pfoser. Towards a flexible and scalable fleet management service. In *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS ’13, pages 79:79–79:84, New York, NY, USA, 2013. ACM.
- [6] A. Efentakis, N. Grivas, G. Lamprianidis, G. Magenschab, and D. Pfoser. Isochrones, Traffic and DEMOgraphics. In *Proc. 21st ACM SIGSPATIAL GIS conf.*, 2013.
- [7] A. Efentakis and G. Lamprianidis. SimpleFleet Deliverable D6.5. SimpleFleet Online Demo [Online]. [http://www.simplefleet.eu/?page\\_id=84](http://www.simplefleet.eu/?page_id=84), 2013.
- [8] Google. The Directions API [Online]. <https://developers.google.com/maps/documentation/directions/>.
- [9] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84(406):502–516, 1989.
- [10] R. Herring, P. Abbeel, A. Hofleitner, and A. Bayen. Estimating arterial traffic conditions using sparse probe data. *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems, September 19-22, Madeira Island, Portugal*, pages 929–936, 2010.
- [11] L. Kabrt. Travel Time Analysis. <http://code.google.com/p/traveltimeanalysis/source/browse>, 2010.
- [12] Laboratory for Software Technology, Computer Science Department, ETH Zurich. Realistic Vehicular Traces [Online]. <http://www.lst.inf.ethz.ch/research/ad-hoc/car-traces/index.html#traces>, 2011.
- [13] X. Liu, F. Lu, H. Zhang, and P. Qiu. Estimating beijing’s travel delays at intersections with floating car data. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS ’12, pages 14–19, New York, NY, USA, 2012. ACM.
- [14] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate gps trajectories. In *Proc. 17th ACM SIGSPATIAL GIS conf.*, GIS ’09, pages 352–361, New York, NY, USA, 2009. ACM.
- [15] Microsoft Research. T-Drive trajectory data sample [Online]. <http://research.microsoft.com/apps/pubs/?id=152883>, 2011.
- [16] O. A. Nielsen, R. D. Frederiksen, and N. Simonsen. Using expert system rules to establish data for intersections and turns in road networks. *International Transactions in Operational Research*, 5(6):569 – 581, 1998.
- [17] OpenStreetMap. Stats - OpenStreetMap wiki [Online]. [http://wiki.openstreetmap.org/wiki/Stats#OpenStreetMap\\_Statistics\\_Available](http://wiki.openstreetmap.org/wiki/Stats#OpenStreetMap_Statistics_Available), 2011.
- [18] OpenStreetMap. [Online]. <http://www.openstreetmap.org/>, 2013.
- [19] OpenStreetMap. Relation:restriction [Online]. <http://wiki.openstreetmap.org/wiki/Relation:restriction>, 2013.
- [20] D. Pfoser and C. S. Jensen. Capturing the uncertainty of moving-object representations. In *Proceedings of the 6th International Symposium on Advances in Spatial Databases*, SSD ’99, pages 111–132, London, UK, UK, 1999.
- [21] QGIS. A Free and Open Source Geographic Information System [Online]. <http://www.qgis.org/>.
- [22] SimpleFleet. Democratizing Fleet Management [Online]. <http://www.simplefleet.eu>, 2013.
- [23] L. Sun. *An approach for intersection delay estimate based on floating vehicles*. Dissertation for Master Degree. Beijing: Beijing University of Technology(in Chinese), 2007.
- [24] F. Viti and H. J. van Zuylen. Modeling queues at signalized intersections. *Transportation Research Record: Journal of the Transportation Research Board*, (1883):68–77, 2004.
- [25] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proc. 18th SSSDBM conf.*, pages 379–388, 2006.
- [26] S. Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.
- [27] H. Zhang, F. Lu, L. Zhou, and Y. Duan. Computing turn delay in city road network with gps collected trajectories. In *Proceedings of the 2011 International Workshop on Trajectory Data Mining and Analysis*, TDMA ’11, pages 45–52, New York, NY, USA, 2011. ACM.
- [28] M. Zhao and X. Li. Deriving average delay of traffic flow around intersections from vehicle trajectory data. *Frontiers of Earth Science*, 7(1):28–33, 2013.
- [29] Y. Zheng and X. Zhou, editors. *Computing with Spatial Trajectories*. Springer, 2011.

# Big data analytics for smart mobility: a case study

Roberto Trasarti<sup>1</sup>

Barbara Furletti<sup>1</sup>

Lorenzo Gabrielli<sup>1</sup>

Mirco Nanni<sup>1</sup>

Dino Pedreschi<sup>1,2</sup>

<sup>1</sup> KDD Lab - ISTI - CNR  
Pisa, Italy  
name.surname@isti.cnr.it

<sup>2</sup> University of Pisa  
Pisa, Italy  
pedre@di.unipi.it

## 1. APPLICATION SCENARIO

This paper presents a real case study where several mobility data sources are collected in an urban context, integrated and analyzed in order to answer a set of key questions about mobility. The study of the human mobility is a very sensitive topic for both public transport (PT) companies and local administrations. This work is a contribution in the understanding of some aspects of the mobility in Cosenza, a town in the South of Italy, and the realization of corresponding services in order to answer to the following questions identified in collaboration with the PT experts.

**Question 1:** How is PT able to substitute private mobility? The objective is to compare public and private mobility to verify the capability of PT to satisfy the user mobility needs.

**Question 2:** How different zones of the city are reachable using PT? This question focuses on understanding how much different zones of the city are served by PT considering different times of the day.

**Question 3:** Are there usual time deviations between real travel times and official time tables? We want to verify if usual time deviations between real travel times and official time tables exist highlighting chronic delays in the service.

**Question 4:** Can we spot visitors and commuters by their behavior? We aim at identifying important categories of people estimating their segmentation in order to evaluate the corresponding demand of services.

For this case study we use data from Cosenza area: a GSM dataset <sup>1</sup>, a GPS dataset <sup>2</sup>, and data from the PT system <sup>3</sup>. GSM data contains 25 mln of phone calls made by about 350K distinct users from 15 October to 9 November 2012. GPS dataset contains about 1.5 mln of private vehicle tracks gathered in February-March and July-August 2012, while PT data consist of a set of GPS logs obtained by the on-

<sup>1</sup>Wind Telecom S.p.a <http://www.wind.it/>

<sup>2</sup>Octotelematics S.p.a. <http://www.octotelematics.com/>

<sup>3</sup>Amaco S.p.a. <http://www.amaco.it/>

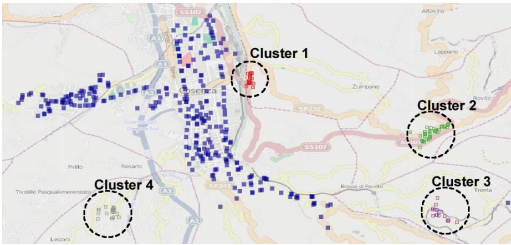
board tracking system of the Cosenza's PT and the PT official time table containing the scheduled times of the arrival of the buses at their stops.

## 2. METHODOLOGY AND RESULTS

To answer the questions posed by the PT manager we developed and implemented a set of methodologies and processes, and we integrated the corresponding services in M-Atlas [3], a larger mobility data analysis framework developed in our laboratory.

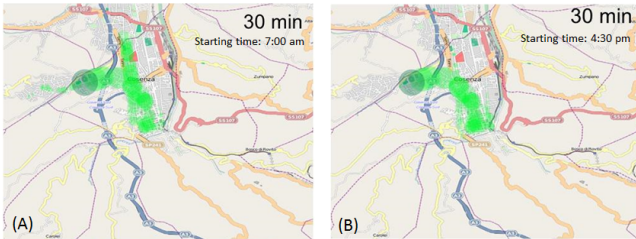
For Question 1 we study the PT capabilities to replace the private mobility in a city. We use the GPS logs of the buses, a *real* time table computed starting from the real buses movements, and the GPS tracks of the private vehicles. We map the PT system to a spatio-temporal network, where nodes are bus stops labeled with name and position, while edges are the connections labeled with origin-destination stops and timestamp. Then, we map the GPS tracks on the PT network and we compute the shortest way to satisfy the users' mobility using an agent-based algorithm that simulates the human mobility in a network [1]. To evaluate the efficiency of the PT we compute the percentage of travels satisfied by the public transport considering a temporal and spatial tolerance (*Coverage*), and the distribution of delays accumulated by the user using the PT instead of the car (*Distribution of time deviations*). Using a maximum *walking distance* of 2 km and applying a temporal constraint of 1 hour as maximum delay, we obtain that the percentage of the user's car travels fully made by using PTs without taking more than 1 hour of extra time is 24%. If we further investigate the delay of the PTs travels w.r.t. the car ones, we find that the delay distribution is affected by the seasonality: in summer the average delay is 29 minutes (with a variance of 26), while in winter is 16 minutes (with a variance of 15). Going back to the trajectory data and extracting the starting points of the users which are not served by the public transport, we can discover which areas are disconnected from the network. By using a clustering algorithm on the starting points of GPS tracks that are not fully covered by the PT we identify two peripheral areas, one industrial and one residential, that are not reached by the bus service (Fig. 1). This result suggests the introduction of new lines or the addition of new bus stops to an existing line passing near those areas. This service is very effective in discovering the real needs of the population and how the network

can handle them, and the analysis may highlight potential customers which can be served by the public transport and therefore good candidates for specific marketing campaign.



**Figure 1: Areas that are not served by the PT service (blue dots are bus stops).**

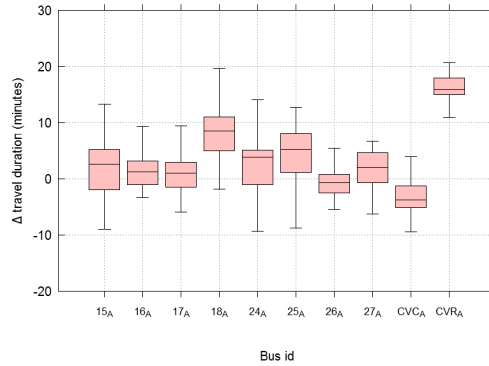
For Question 2 we try to understand which areas of the city can be reached starting from a specific bus stop at a specific time of the day, having a fixed amount of time on the PT network. As a result we find that particular areas of the city can be reached by the PTs in a fixed amount of time only in certain time slots, as shown in Fig. 2. This service allows the PT manager to add lines or modify the bus schedule for analyzing the impact of his choice in the PT system.



**Figure 2: The reachability of the city starting from the darker point at 7:00 am (A), and 4:30 (B) and having 30' of time available.**

We answer to Question 3 by computing the differences between the expected duration of the bus as stated in the official time table and the one inferred by the bus log. Fig 3 shows that almost all the buses are late in a range  $[10, -10]$  min. except for bus CVR A which has an average delay of 17 min. with a very small variance. This kind of information is very useful to spot problems in the buses management, i.e. to improve the service or to highlight too strict schedules which can't be respected by the buses in reality. As a result we draw a complete map of the typical behaviors of the buses and we identify the most critical lines. The last consideration is about the buses which makes the travel faster than expected, these are clearly buses which try to reduce the delay accumulated in previous travels. This behavior is harmful and makes the time table unreliable.

To answer to Question 4 we apply the analytic process described in [2] which analyzes the calling behavior of the users in order to classify them into three categories: *Resident*, *Commuters* and *Visitors*. People that appear only once (i.e. that make only one call in all the period of observation) be-



**Figure 3: Official schedule vs. inferred one.**

fore disappearing are separately classified in the *In Transit* category. We obtain the following segmentation: 23.12% of Residents, 14.56% of Commuters, 26.45% of Visitors, and 28.74% of In transit. The 7.13% are *unclassified* due to their unclear profile. GSM data are a good proxy to compute people presence in a territory with a certain regularity and with an economic convenience because survey campaigns are expensive and time consuming. Our indicator based on GSM data helps to manage and re-arrange the resources and services w.r.t. the user demand.

The collaboration with the public administration helped us to identify several key questions concerning the mobility needs and the transportation offers. By exploiting the peculiarities of the different data sources that were available in the application context we answered producing a set of analyses and implementing a set of services for extracting useful and new knowledge. The results have been tested on the field, allowing a continuously monitoring of the general status and health conditions of the urban traffic, in terms of impact of PT, actual mobility demand, and mobility profiles of citizens living in the area. We consider this as a preliminary work towards the definition of a sort of dashboard for a mobility manager composed of a set of end-user services and indexes to evaluate the transport system of a city.

### 3. ACKNOWLEDGMENTS

This work has been partially funded by the European Union under the National Operational Program Research and Competitiveness 2007-2013: Project Tetris n. PON1\_00451; and under the FP7-ICT Program: Project DataSim n. FP7-ICT-270833.

### 4. REFERENCES

- [1] F. Pinelli et al. *Space and time-dependant bus accessibility: a case study in Rome*. Proc. of the 12th IEEE Conf. on ITS, 2009.
- [2] B. Furletti et al., *Analysis of GSM Calls Data for Understanding User Mobility Behavior*. IEEE Big Data, 2013
- [3] F. Giannotti et al., *Unveiling the complexity of human mobility by querying and mining massive trajectory data*. VLDB Journal Special issue on Data Management for Mobile Services (2011).

# Smart Applications for Smart City: a Contribution to Innovation

Simona Citrigno  
Centro di competenza ICT-SUD  
87036 Rende CS  
simona.citrigno@cc-ict-sud.it

Francesco Lupia  
Università della Calabria  
87036 Rende CS  
lupia@dimes.unical.it

Sabrina Graziano  
OKT srl  
87036 Rende CS  
sabrina.graziano@okt-srl.com

Domenico Saccà  
Università della Calabria  
87036 Rende CS  
sacca@unical.it

## ABSTRACT

Main research activities and results of the project “TETRis – TETRA Innovative Open Source Services” are described that are aimed at enabling innovative services for Smart City/Smart Territory by means of technological tools and intelligent platforms for collecting, representing, managing and exploiting data and information gathered from sensors and devices deployed in the territory. Technological tools and intelligent platforms are integrated into two smart environments for monitoring of respectively mobility and environmental resources and for providing advanced services to citizens as well as to urban operators. The general architecture and goals of the two smart environments are illustrated and some insights on the prototype for the mobility monitoring environment are reported.

## Categories and Subject Descriptors

K.6.1 [Management of Computing and Information Systems]: Software and People Management – *strategic information systems planning, systems analysis and design, systems development.*

## General Terms

Management, Design, Experimentation, Human Factors, Measurement.

## Keywords

Urban Monitoring, Urban Mobility, Intelligent Platforms

## 1. INTRODUCTION

The activities described in this paper are related to the design and prototypal implementation of innovative services aimed at an intelligent management of an urban territory for novel smart city application scenarios. Within these activities, a number of solutions and advanced technology platforms have been identified

that enable the various entities operating in the area of interest (municipalities, provinces, regions, universities, etc.), as well as citizens and urban operators, to effectively cooperate for an efficient usage of urban resources.

The innovation scenarios and solutions described in this paper have been realized within the project PON 2007 2013 - Research and Competitiveness "TETRis - TETRA Innovative Open Source Services" according to reference general frame of "Internet of Things" for supporting Smart City/Smart Territory [1], in which the acquisition of data by objects is applied to large territorial areas by exploiting the widespread availability of communication networks [2]. The collected data, properly enhanced and enriched, foster innovative services oriented to the production and exchange of knowledge among the different actors interconnected in urban and regional networks. The development of these services has been realized through a smart environment enabling the cooperation of smart devices and objects as well as of operators and users of the services themselves.

Two smart environments have been designed to handle two relevant smart city application scenarios: (i) *Urban Mobility Monitoring* and (ii) *Territory Monitoring, Control and Maintenance*. The general architecture and the goals of the two environments are described in Section 2. In addition some insights on the design and prototype implementation of the smart environments on Urban Mobility Monitoring are reported in Section 3. Conclusions are withdrawn in Section 4.

## 2. TWO SMART ENVIRONMENTS FOR SMART CITY

Each of the two smart environments has been designed as a knowledge-based digital/physical system that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in stationary and mobile smart objects with embedded intelligence and in smart-phones, and connected through a continuous network. The specific goals of the two smart environments are described next:

*Smart Environment for Urban Mobility Monitoring:* it concerns the implementation of a model for the detection of mobility problems in urban areas through the use of stationary smart objects deployed in the territory and mobile ones installed in public transportation buses. Data from smart objects are

collected and aggregated into a data warehouse feeding a Mobility Intelligence platform defined through the design of innovative techniques of space-temporal data analysis and mining of complex data, including trajectories [3]. The model also includes amenities to deliver services to operators and citizens through the use of mobile devices. The environment has been experimented in the town of Cosenza in Southern Italy.

*Smart Environment for Territory Monitoring, Control and Maintenance:* it is based on networks of physical sensors connected to smart objects as well as of "social" sensors (smart-phones driven by citizens and urban operators) to detect the status of the territory in real time. These so-collected data are stored and aggregated into a data warehouse feeding a Territory Intelligence platform, which enables the extraction and processing of knowledge for monitoring the territory. Other important components of the environment are contextual applications and web portals for citizens and operators. The environment has been experimented in the town of Rende in Southern Italy.

### 3. A PROTOTYPE FOR URBAN MOBILITY MINING

The smart environment for Urban Mobility Monitoring includes a prototype for urban mobility mining based on advanced data mining techniques that are used into two tasks: (1) a *pre-processing task* (i.e., extracting data from spatio-temporal databases and converting them into a process log) and (2) a *mining task* (i.e., building a process model for a given input log).

During the pre-processing task, a number of possible anomalies are detected for which it is necessary to perform an in-depth analysis to discover causal dependencies that hold over them. To this end, starting from an input log, the mining task builds a dependency graph (e.g., a Petri Net) that represents the background knowledge in terms of *precedence constraints* over the causal dependencies that, in many cases, are available to the analyst. This is particularly useful in order to circumvent the problems emerging when frequent incompleteness of logs. Note that "traditional" process discovery methods do not provide any support to deal with prior knowledge.

A relational database is used to initially store data on bus mobility in the urban area of Cosenza. Then these data are filtered and reorganized in the form of specific sequences of events so that they can be seen as traces of a process. In more detail, each tuple of the database is a triple of the form  $\langle x, s, t \rangle$  where  $x$  is an object identifier,  $s$  is spatial position and  $t$  is a timestamp (i.e., the time at which the event has been recorded). Then this set of spatio-temporal tuples are converted into a set of transition instances consisting of three properties: a route identifier, a starting instant of the instance of transition and a measure associated with the instance of transition (e.g., travel time).

Two consecutive tuples (w.r.t. the temporal order)  $R_i = (x, s_i, t_i)$  and  $R_j = (x, s_{i+1}, t_{i+1})$  of the same object  $x$  are seen as an instance of transition for the route  $s_i \rightarrow s_j$  if the difference  $t_{i+1} - t_i$  is below a certain threshold (e.g. events that have occurred in the same day). For example, in the specific case of a tuple for urban transport system mobility, the transition  $s_i \rightarrow s_j$  corresponds to the pair of stops  $s_i$  and  $s_j$  for which there exists a line that connects  $s_i$  and  $s_j$ .

Next step is to generate a set of classes/intervals, according to which anomalies can be classified and events can be thereafter clustered according to their class ids. Basically, each class specifies how the measure associated with the event/activity,

exceeded the average value associated with the aggregate view of the same event.

Finally, a cluster and a time interval duration attribute are chosen that will help to generate a *trace id* to label a set of unexpected related events forming a trace. In particular, two events  $E_i$  and  $E_j$  belong to the same trace if:

- both events occurred in a "fixed" time interval;
- there exists another event  $E_z$  which connects  $E_i$  and  $E_j$ .

Given the input log generated as described above, classical mining techniques have been applied. It is to be noticed that some of the process models presented two types of conceptual problems:

- congestion models (see Figure 1) on which causality between two routes, one after another, follow driving direction (traffic propagates causally in the opposite driving direction w.r.t. the origin of the obstruction).
- congestion models (see Figure 2) with causality between routes that are geographically distant.

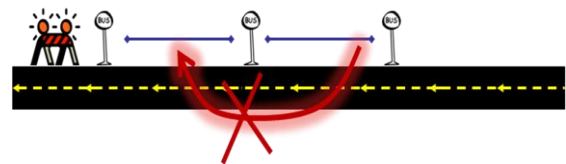


Figure 1. Inverted causal dependency example



Figure 2. Causal dependency between distant routes

Finally using ad-hoc mining techniques developed in the research activities of the TETRIS project, it was possible to encode the domain knowledge, thus restricting flow models to those that do not present the above-mentioned problems.

### 4. CONCLUSION

Some of the activities and results of the TETRIS project have been illustrated that concern the definition and experimentation of innovative solutions for monitoring urban contexts according to the emerging integrated strategic vision of the Smart City and for providing ubiquitous services to both citizens and urban operators.

### 5. REFERENCES

- [1] Komninos N., Schaffers H., Pallot M., "Developing a Policy Roadmap for Smart Cities and the Future Internet", eChallenges e-2011 Conference
- [2] European Commission, "Smart Cities and Communities – Support for a better future", 2013: <http://ec.europa.eu/eip/smartcities/>
- [3] Giannotti F., Nanni M., Pedreschi D., Pinelli F., Renso C., Rinzivillo S., Trasarti R.: Unveiling the complexity of human mobility by querying and mining massive trajectory data. VLDB Journal Special issue on Data Management for Mobile Services (2011).

# Analysis of Relationships Between Road Traffic Volumes and Weather: Exploring Spatial Variation

Jaakko Rantala

Department of Real Estate, Planning and  
Geoinformatics  
Aalto University  
Espoo, Finland

jaakko.rantala@aalto.fi

James Culley

Department of Real Estate, Planning and  
Geoinformatics  
Aalto University  
Espoo, Finland

james.culley@aalto.fi

## ABSTRACT

Weather is known to have a strong effect on traffic volumes. In this paper, we suggest a spatial approach to the modelling of traffic volumes. The relationship between weather variables and traffic volume is first modelled at a global level in a regional city centre in Finland. As strong a spatial dependency is found between the variables in the model, spatial variation is incorporated into the model. This local approach provides a more accurate model, as well as new insights into the data.

## Categories and Subject Descriptors

G.3 [Probability and Statistics]: Correlation and regression analysis

## General Terms

Measurement, Theory.

## Keywords

Weather, traffic volume, spatial modelling.

## 1. INTRODUCTION AND BACKGROUND

The modelling of traffic flows and volumes has been an important topic of research in varied fields and has previously been studied with the use of a raft of methods and techniques [3, 5]. One element that is lacking in previous studies into how the weather affects traffic volumes is the incorporation of spatial variation.

Previous studies that have examined the influence of weather variables upon traffic volumes have assumed that any relationships that are determined hold constant for the whole study area. In this paper we explore whether this assumption is valid or whether there are intrinsic differences in the relationships affecting traffic volumes which can be identified at disparate locations across the study area.

## 2. DATA

Traffic volume data was received from the city of Oulu, Finland. The data is for the whole of 2012 and consists of the daily totals

of traffic volumes from 59 crossroads as measured by sensors located at each of them.

Weather data was received from the Finnish Meteorological Institute. From the data measured every ten minutes we obtained variables relating to the daily average temperature, amount of precipitation, and average road friction.

## 3. MODEL DESCRIPTION

### 3.1 Ordinary least squares

Ordinary least squares (OLS) regression is applied to the data first. This is a global model that results in a single estimation of each parameter for the entire study area [2]. The classic OLS regression model is written as

$$Y_i = b_0 + \sum_k b_k X_{ik} + \varepsilon_i$$

where  $Y$  is the dependent variable,  $X_k$  is the  $k$ th explanatory variable, and  $b$  are the regression coefficients to be estimated by the model from the observed data.

In the model we adopted, the normalised traffic volume is the dependent variable. The explanatory variables are the days of the week, school and public holidays, the summer period, and weather variables for temperature, precipitation, and friction.

### 3.2 Local model

The most common method for modelling spatially non-stationary relationships is through geographically weighted regression (GWR) [1]. Whilst we would have liked to use GWR for our study, the data does not lend itself favourably to an analysis, with 59 crossroads locations, with 365 events at each location.

Therefore we propose two solutions to this problem. First, we will run a separate regression at each location to study the spatial autocorrelation. Second, we will use a simple form of GWR: A regression model is calibrated on all data that lies within the distance  $d$  of the regression point, a crossroads, and the process is repeated at all the regression points. We shall term moving window regression MWR in this paper.

## 4. RESULTS

### 4.1 Global model

All the coefficients in the global regression model are statistically significantly different from zero. The  $R^2$  value, which is a goodness-of-fit statistic that shows how much variation in the traffic volume is explained by the model, is 0.76.

The results show that on days when there was precipitation there was 2% more traffic. A rise in temperature of 10 degrees Celsius



results in 0.6% more traffic and the maximum possible increase in friction equals a 6% increase in traffic compared to the median in the model.

## 4.2 Local model

To study if spatial dependency exists in the data, first, regression models are created for all the crossroads individually. Moran's I values [4] are then calculated for the coefficients to study the spatial autocorrelation.

All the Moran's I values for the coefficients show a positive spatial autocorrelation. Thus the relationship between the weather variables (as well as the time variables) and the amount of traffic varies spatially. This indicates that when studying the effect of these variables upon traffic volumes we need to take the location into account.

For the MWR we need to determine a suitable bandwidth distance  $d$  to use in the model. Different values were tested and the results presented here were calculated with the value of 1500 metres.

The coefficients from the local models can be shown on a map to study how the variables explain the traffic volumes in different areas.

Figure 1 presents the local coefficients for the precipitation variable. It shows that in the central area and northern part of Oulu the coefficients are bigger than in the other areas on the outskirts, which means that in those areas the traffic increase on days when there is precipitation is bigger, up to almost 4%. One reason for this may be that cyclists in the Oulu area switch to cars when it rains or snows and there are more cyclists in the centre than in the outskirts. The local coefficients for the other weather variables show a similar pattern.

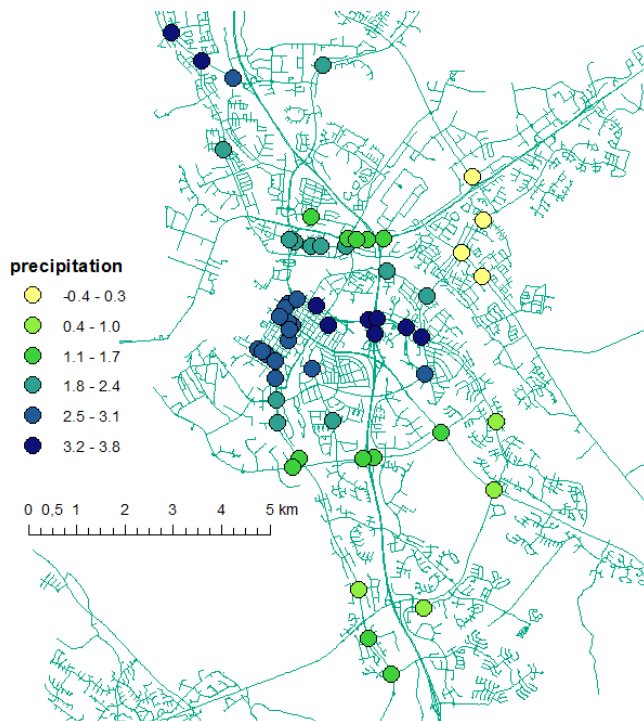


Figure 1. Local coefficients for the precipitation variable.

The temporal variables can be studied in a similar manner to reveal the effects of different days of the week, as well as holidays. For example, on weekdays the increase in traffic is at its biggest south of the centre. One reason for this is that the area has lots of industrial, commercial, and office buildings.

The goodness of fit can also be studied locally by plotting the local  $R^2$  values. The values are better than for the global model, except in the central areas. Thus it seems that in the central area there are possibly explanatory variables missing from the model.

## 5. CONCLUSION AND DISCUSSION

This study strongly indicates that modelling the correlation between traffic volumes and weather variables needs a spatial approach. This is because the relationship between the variables depends on location. The centre of the city is especially different from other areas.

Possibly the biggest constraint for the research is the time span for the data. As only the data from one year is used, seasonal variation is not analysed and long-term trends cannot be seen.

The results help when predicting traffic volumes in Oulu. As the amount of traffic affects travelling time, this can be used for navigational purposes. In order to create a better spatial model, more sensors are needed outside the city centre.

Because the concept of spatially non-stationary relationships is scarce in the traffic modelling literature, we decided to approach the topic with a simple method, suitable for preliminary investigation. The purpose of adopting this technique was to identify whether there was any evidence to support the concept of non-stationary spatial relationships in the traffic modelling. The next stage in this research will be to determine a more suitable technique to investigate the apparent spatial aspects.

## 6. ACKNOWLEDGEMENTS

This research was performed as a part of the Data to Intelligence (D2I) project funded by Tekes, Finland. We are also grateful to the city of Oulu and the Finnish Meteorological Institute.

## 7. REFERENCES

- [1] Brunson, C., Fotheringham, A. and Charlton, C. 1996. Geographically weighted regression: A method for exploring spatial nonstationarity. *Geogr. Anal.* 28, 4 (Oct. 1996), 281-298.
- [2] Fotheringham, A., Brunson, C. & Charlton, M. 2002. *Geographically Weighted Regression: the analysis of spatially varying relationships*. Wiley, Chichester.
- [3] Koetse, M. J. and Rietveld, P. 2009. The impact of climate change and weather on transport: An overview of empirical findings. *Transport. Res. D-Tr. E.* 14, 3 (May 2009), 205-221.
- [4] Moran, P. A. P. 1950. A test for the serial independence of residuals. *Biometrika.* 37, 1 (June 1950), 178-181.
- [5] Rakha, H., Farzaneh, M., Arafteh, M., Hranac, R., Sterzin, E., and Krechmer, D. 2007. *Empirical studies on traffic flow in inclement weather*. Technical report. Virginia Tech Transportation Institute. <http://www.mautc.psu.edu/docs/VPI-2005-01.pdf>.

# SiCi Explorer: Situation Monitoring of Cities in Social Media Streaming Data

Andreas Weiler, Michael Grossniklaus, and Marc H. Scholl  
Database and Information Systems Group, University of Konstanz  
Box D188, 78457 Konstanz, Germany  
firstname.lastname@uni-konstanz.de

## ABSTRACT

The continuous growth of social networks and the active use of social media services result in massive amounts of user-generated data. More and more people worldwide report and distribute up-to-date information about almost any topic. Therefore, we argue that this kind of data is a good basis to observe ongoing situations in cities as well as related situations from outside about these cities in real-time.

This paper presents a visualization for monitoring the situation (current topics and emotions) in cities and about cities, which is reflected in the live message stream of the social microblogging service *Twitter*, by using continuously updating and with sentiment colored TagClouds.

## 1. INTRODUCTION AND MOTIVATION

The high volume and distribution speed of tweets makes it difficult for users to follow the evolution of topics within the continuous data flow. It is a big challenge to discriminate between normal behavior of the social sensor or unusual and abnormal behavior, which usually is an indicator for an interesting event in the area. However, the amount of useful information in the generated data increases as well. Another advantage of user-generated data is the automatic enrichment of the textual information by geographical information of the user's mobile device. Hereby, it is possible to classify the incoming information as local report or report about a city from outside.

In this paper, we present a visualization for monitoring situations in cities and surrounding areas, which is filtered by geographical coordinates, and the situation, which is reported from outside by filtering for keywords, in the live and continuous streaming data of *Twitter*. Our work presents a compact visualization for time series event data, which supports users to identify interesting data points inside the cities and about these cities from outside. This supports users in following the evolution of topics and emotions of locals in defined geographical areas and also to visualize the evolution of topics and emotions of people about the area. It also

supports the users in differentiating between the situation inside the area and the situation from outside.

## 2. DESIGN

To visualize the evolution of topics and the emotion in and about a city over time, we use different layers of granularity displayed on a map. Each layer is split into two parts. The first part (inside the oval area, filtered by geo-coordinates) reflects the situation inside the area of the city and the second part (inside the rectangle area without the oval area, filtered by keywords) reflects the situation from outside about the city. The TagClouds of both parts are adjusted to the shape of the area, by using a processing library<sup>1</sup>, and so no overlapping terms between the two areas exists. The upper layers are divided into slices, which reflect the sentiment from the underlying layers. Hereby, it is possible to recognize interesting points in time in underlying layers even if the user just observes the top layer.

**Top-Layer:** The lowest granularity layer reflects the topics and emotions in hourly frames. To reflect the sentiment of the middle-layer both areas are split into six slices. These six slices are colored in the corresponding sentiment of the middle-layer.

**Middle-Layer:** The middle granularity layer reflects the situation in ten minute frames. Both areas are split into ten slices, which reflect the corresponding sentiment from 1 minute frames from the bottom-layer.

**Bottom-Layer:** The highest granularity layer reflects the topics and emotions in one minute frames in real-time.

The emotion of a time frame is visualized by using the color dimension. The fill color of the shape signifies the average sentiment (red = negative, green = positive) of the tweets in the data window. The value of the sentiment for a tweet is obtained by using an external library<sup>2</sup>, which analyzes the text of the message and returns sentiment values between  $-5$  (extremely negative) and  $5$  (extremely positive).

## 3. USE CASE

The use case describes the observation of the city *Boston* (filtered for 5 miles around the city center and the keyword "boston") on the day of the 15th April 2013. Figure 1 shows the overview of the whole use case for the hours from 5PM to 9PM. The most important term "marathon" in the first

<sup>1</sup><http://wordcram.org/>

<sup>2</sup><http://sentistrength.wlv.ac.uk/>

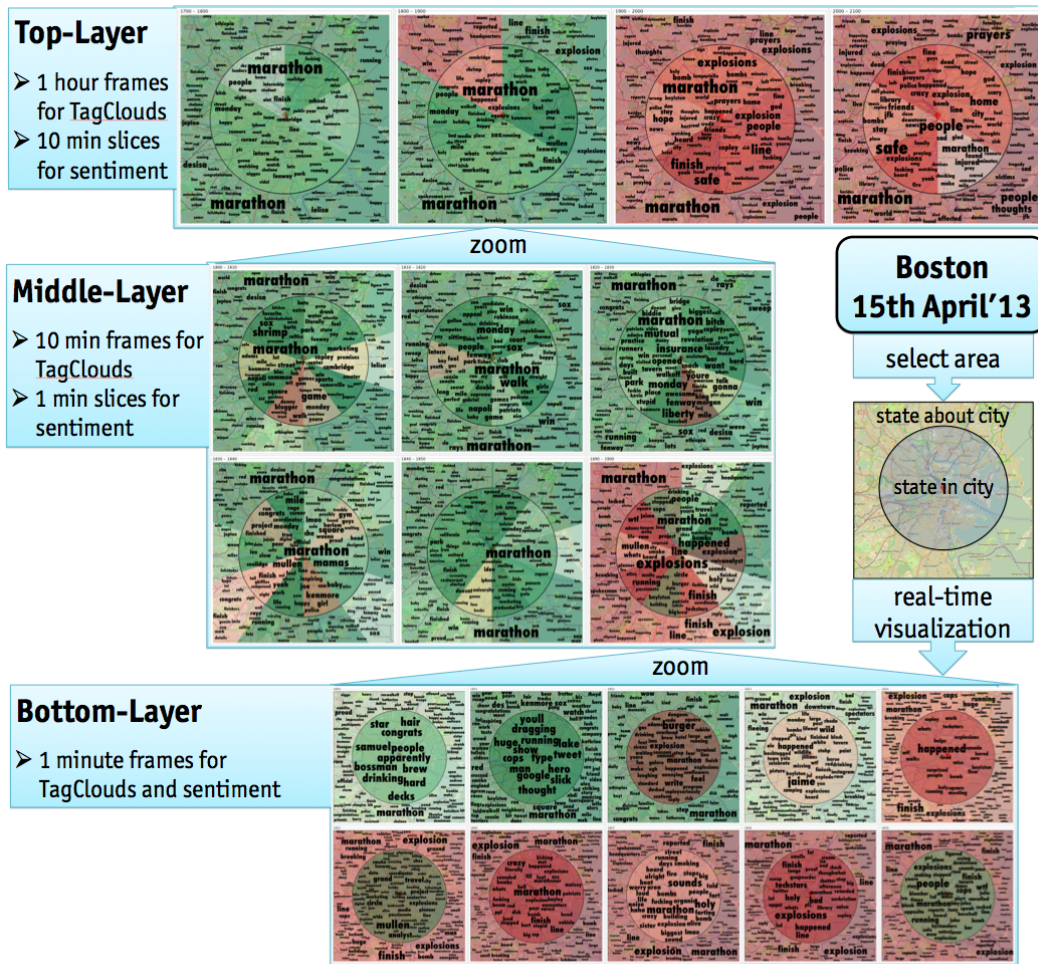


Figure 1: Process of the use case for the city Boston on 15th April 2013, from 5PM to 9PM (UTC).

hour signifies the ongoing sports event in the city and also reflects a very positive emotion for inside the city and from outside. Further indicators for the Marathon are terms like “running”, “finish”, “desisa” (name of the male winner), and “jeptoo” (name of the female winner). The second hour also shows very positive emotion for the first five segments of the hour. However, in the sixth segment, the last ten minutes of the hour, the emotion shifts to negative for inside the area and from the outside. Also, the term “explosion” can be seen in the outside and inside part. By looking at the two following hours, we notice that the negative emotion increases sharply in both areas. Further terms like “bombs”, “prayers”, and “thoughts” indicate that a bad event happened in the city of Boston. An interesting observation is that the terms “prayers” and “thoughts” are more frequent on the outside. Since the second hour shows this abnormal behavior and we are further interested in the evolution of the situation, we zoom into the second hour of the visualization. In this more detailed view, we can see that the term “marathon” is always very frequent, however in the last ten minutes of the hour the terms “explosion”, “finish”, and “line” on the outside and the terms “explosions”, “happened”, and “finish” on the inside reflect the just happened *Boston Marathon Bombings*. After analyzing the middle layer of the visualization, we are

further interested in the situations in the last ten minutes of the second hour. Therefore we zoom into the last ten minutes of the middle-layer, where we can identify that the term “explosion” appears in both areas in the second minute. In contrast, the emotion is still positive at that time, because the event not yet widely spread. We can summarize that by using the visualization the explosion can be identified by terms only two minutes after the event took place. However, the emotion changes only slightly in the first minutes, but then the negative emotion increases and gives a good indication about the extent of the tragedy.

#### 4. CONCLUSIONS

In this paper, we demonstrated a visualization for monitoring the situation (current topics and emotions) within and about cities, which is reflected in the live message stream of the social microblogging service Twitter. Our use case shows that it is possible to visualize the current and past topics and emotions for cities. We can also conclude that the live observation can support local people and news reporters in getting up-to-date information about the state of emotion and topics in and about a city.

# A Cascading Wavelet-Feed Forward Neural Network Approach for Forecasting Traffic Flow

Md. Mostafizur Rahman  
National Institute of  
Informatics  
Tokyo 101-8430, Japan  
mostafizur\_du27@yahoo.com

Atsuhiko Takasu  
National Institute of  
Informatics  
Tokyo 101-8430, Japan  
takasu@nii.ac.jp

Hafiz Md. Hasan Babu  
University of Dhaka  
Dhaka-1000, Bangladesh  
hafizbabu@hotmail.com

## ABSTRACT

Predicting Traffic flow in the busiest cities has become a popular research area in the past decades. The rapid development of intelligent traffic management system attracts the software industry to come up with efficient tools for traffic prediction over the roads. In this study, Discrete Wavelet Transformation (DWT) is employed with Artificial Neural Network (ANN) to forecast the traffic flow over the roads by analyzing loop sensor's data. An Information Theoretic Approach has been extended for choosing the number of nodes in hidden layer of Neural Network for the proposed model. The proposed hybrid model was compared with standard Artificial Neural Network (ANN) model. The forecasted results showed that proposed joined Wavelet and Feed Forward Neural Network (WFFNN) worked much well over the experimental data than ANN model.

## Keywords

Time series, Discrete wavelet transformation, Feed forward neural network, Traffic flow.

## 1. INTRODUCTION

Significant statistical information for both past and near future can be extracted by analyzing time-series data. Traffic congestion, volumes, origins, routes and other road-traffic performance metrics are useful for designing urban traffic control systems and these types of data typically exhibit a periodicity in time. Traffic data can be collected manually or via static sensors such as traffic cameras and loop detectors. In this paper, we analyzed time-series loop sensor data and proposed a model for daily traffic forecasting (traffic condition around the day in specific time-gaps) for next upcoming weeks. Two datasets (Dodgers and TSF generated data) have been used for testing purpose of the proposed model [3, 2]. We analyzed and trained the model with 18 weeks traffic counts (car counts of every 5 minutes in a day) obtained from Dodgers loop sensor dataset and forecasted the traffic condition of upcoming seven weeks. Each day

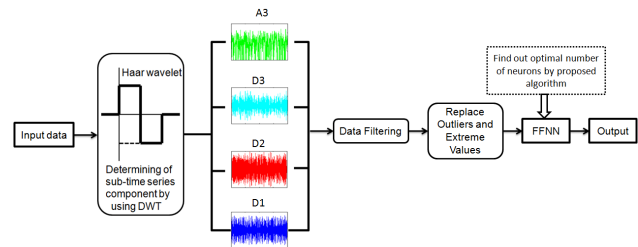


Figure 1: Schematic diagram of the proposed WFFNN model.

contains 288 time-slice predictions. In the similar way, we trained our model with the average velocity of 4000 observations (recorded the average velocity in every five seconds) obtained from TSF generated data and forecasted what will be the next 1000 observations or average velocity of the simulation model. So, the main focus of this paper is analyzing time-series loop sensor traffic data and examine the applicability of wavelet-feed forward neural network based modeling.

In this study, an algorithm has been proposed based on theoretic approach (Jump Method) and it is observed that proposed algorithm works well for choosing the best number of neurons for a neural network. Wavelet-Neural Network approach can be useful for different application fields. Here, we used the technique in analyzing traffic data and added data filtering for different sub-signals before summing up the sub-signals and an algorithm for determining the optimal number of neurons on the training period of the proposed model.

Wavelets analysis is localized in both time and frequency while the Fourier transform is only localized in frequency. So, combine model of wavelets and neural network has been used for analyzing time series data and enhance the prediction performance of neural networks. This paper deals with loop sensor's data and forecasts the traffic congestion by counting the cars or average velocity of vehicles. The main purpose was employing wavelets to analyze large loop sensor time-series traffic information and extract the traffic condition over time.

## 2. MODELING STRATEGY OF WFFNN

This study employed Discrete Wavelet Transformation cascaded with Feed Forward Neural Network. Here, Mallat Discrete Wavelet Transformation has been adopted. Mal-



**Table 1: Testing results of Dodgers loop sensor dataset.**

Number of Neurons	Proposed WFFNN Model			ANN		
	MSE	RMSE	MAPE(%)	MSE	RMSE	MAPE(%)
7	8.1524	2.8552	3.9605	37.0948	6.0906	20.9647
8	7.0129	2.6482	3.7494	35.7893	5.9824	16.7499
9	7.1152	2.6674	4.0770	28.7240	5.3594	11.4475
10	7.1354	2.6712	4.1003	32.2587	5.6797	19.8356

**Table 2: Testing results of TSF generated data.**

Number of Neurons	Proposed WFFNN Model			ANN		
	MSE	RMSE	MAPE(%)	MSE	RMSE	MAPE(%)
4	5.1040	2.2592	2.2678	23.3926	4.8365	12.2045
5	4.9098	2.2158	1.9827	24.2849	4.9279	14.3829
6	4.9194	2.2180	2.2012	20.3916	4.5158	9.5893
7	5.0923	2.2566	2.2156	19.2938	4.3924	6.1047

lat’s algorithm can be expressed as follows [5]:

$$\begin{aligned} a_{j+1} &= Qa_j \text{ where } j=0,1,2,\dots,J \\ d_{j+1} &= Ga_j \text{ where } j=0,1,2,\dots,J \end{aligned} \quad (1)$$

Here, in the equation Q and G are low pass filter and high pass filter respectively. If  $a_o$  represents the original time series T, then T can be decomposed to  $d_1, d_2, d_3, \dots, d_j$  and  $a_j$ , where J is the scale.  $a_j$  and  $d_j$  are the approximated coefficients and detail coefficients of original time series.

Firstly, the input data ( $C_t$ , car counts in every five minutes in Dodgers loop sensor data and  $V_t$ , average velocity in TSF generated data) have been decomposed into a certain number of sub-time series components by DWT. Input time series first decomposed into approximation and detail coefficients. In this way decomposition process is iterated and successive approximation signals being decomposed in turn.

Best results for two data sets have been obtained by three decomposition level. Input data have been decomposed with Haar wavelet function and Daubechies wavelet function. Consequently,  $D_1, D_2, D_3$  were detail time series and  $A_3$  was approximation time series. To get more accurate results we employed Interquartile Range (IQR) for clustering and finding the outliers and extreme values [1]. In this study, outliers have been replaced with mean values of respective data sets. Extreme values have been normalized.

In this study, the wavelets sub series  $\{D_1, D_2, D_3, A_3\}$ , were summed together after removing insignificant coefficients, which is similar as [4], and feed to a Feed Forward Neural Network at time  $t$  and original time series at time  $(t + t_f)$  are outputs of FFNN, where  $t_f$  is the length of time to forecast. Predicted output sets were later compared with the actual values. Number of neurons of the hidden layer of FFNN has been chosen by proposed algorithm (Algorithm 1) and it is observed that proposed algorithm was very effective for the experimental datasets. The schematic diagram of proposed model is shown in figure 1.

### 3. EXPERIMENTAL RESULTS

To compare the efficiency of the proposed model, we generated ANN model and tested with the same inputs. Performance indices were presented in Table 1 for Dodgers loop sensor data and Table 2 for TSF generated data. It is clear from Table 1 and Table 2 that MSE (Mean Square Error), RMSE (Root Mean Square Error) and MAPE (Mean Abso-

### Algorithm 1 Extended Jump Method for Finding Neurons

The MSE (Mean Square Error)  $d_k$  equals the variance of residuals generated by fitting the neural network model with k nodes.

**for** k=1 **to**  $k_{max}$  **do**

    Calculate MSE,  $d_k$

**end for**

**for** k=1 **to**  $k_{max}$  **do**

    Calculate priority factor  $P_k$  for  $d_k$  (Assigning priority value or weight based on MSE value).

**end for**

Choose two positive numbers  $v > 0$ , called the transformation power and  $b > 0$ , called priority bias factor.

**for** k=1 **to** n **do**

**if** k=1 **then**

$$J_k = b.P_k.d^{-v}$$

**else**

$$J_k = b.P_k.d_k^{-v} - d_{k-1}^{-v}$$

**end if**

**end for**

The best number of nodes is the number k lies between two highest picks such that Node number of  $(J_{second\_highest}) \leq k \leq$  Node number of  $(J_{top\_highest})$ .

lute Percentage Error) values for proposed WFFNN (results of using Haar wavelet have been presented in two tables) model are much better than ANN model.

Although the forecasting accuracy of ANN model in respect of MAPE is 11.4475% for Dodgers loop sensor data and 6.1047% for TSF generated data. Best result for the Dodgers loop sensor data by using WFFNN model found at neuron number 8 with Haar wavelet and MAPE is 3.7494%. Best result for TSF generated data came up with Haar wavelet function by using 5 neurons in the hidden layer and there MAPE value is 1.9827%, which is very much acceptable.

### 4. CONCLUSION

This paper presented a hybrid prediction approach and it’s application based on discrete wavelet transformation and feed-forward neural network. Our proposed model can provide good accuracy in predicting real time traffic and works well over loop detector or sensor data. It can be used as a useful tool for Advanced Intelligent Transport Systems.

### 5. REFERENCES

- [1] F. M. Dekking. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer, 2005.
- [2] P. Gora. Traffic simulation framework. In *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*, pages 345–349. IEEE, 2012.
- [3] J. Hutchins. *Freeway Performance Measurement System (PeMS)*, “<http://pems.eecs.berkeley.edu>”.
- [4] O. Kisi and M. Cimen. A wavelet-support vector machine conjunction model for monthly streamflow forecasting. *Journal of Hydrology*, 399(1):132–140, 2011.
- [5] S. G. Mallat. A theory for multiresolution signal decomposition: the wavelet representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(7):674–693, 1989.

# Combining a Gauss-Markov model and Gaussian process for traffic prediction in Dublin city center

François Schnitzler  
Technion  
Fishbach Building  
32000 Haifa, Israel  
francois@ee.technion.ac.il

Shie Mannor  
Technion  
Fishbach Building  
32000 Haifa, Israel  
shie@ee.technion.ac.il

Thomas Liebig  
TU Dortmund University  
Artificial Intelligence Group  
Dortmund, Germany  
thomas.liebig@tu-dortmund.de

Katharina Morik  
TU Dortmund University  
Artificial Intelligence Group  
Dortmund, Germany  
katharina.morik@tu-dortmund.de

## ABSTRACT

We consider a city where induction-based vehicle count sensors are installed at some, but not all street junctions. Each sensor regularly outputs a count and a saturation value. We first use a discrete time *Gauss-Markov model* based on *historical data* to predict the evolution of these saturation values, and then a *Gaussian Process* derived from the *street graph* to extend these predictions to all junctions. We construct this model based on *real data collected in Dublin city*.

## Categories and Subject Descriptors

G.3 [Probability and Statistics]: Markov processes, multivariate statistics, stochastic processes, time series analysis; I.2.6 [Artificial Intelligence]: Learning—parameter learning; J.7 [Computer in Other Systems]: Real time

## Keywords

traffic prediction, Gaussian Process, Gauss-Markov, autoregressive, smart cities, time series, spatio-temporal

## 1. INTRODUCTION

In the Greater Dublin Area, 750 (4%) junctions are covered by one or several SCATS (Sydney Co-ordinated Adaptive Traffic System) vehicle count sensors. Our goal is to provide estimates of the saturation at each junction, for the current and future times, whereas our previous work [1] only did so for each junction at the current time.

High traffic saturation (cars/km) co-occurs with low traffic flux (cars/hour) and is an indicator for congestions [3].

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

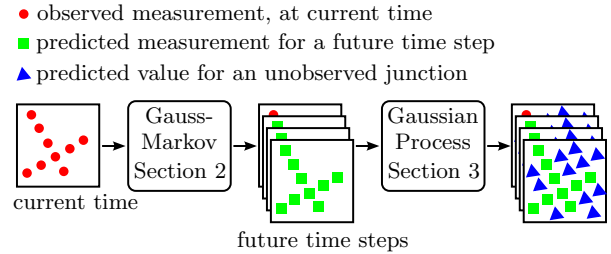


Figure 1: Future measurements are estimated by a Gauss-Markov process (Section 2). Estimates for junctions without sensors, are provided by a Gaussian Process (Section 3).

Our work can be used for online signaling and trip planning.

The urban street network is a graph  $(V, E)$ , where the vertices  $V$  are the junctions and the edges  $E$  the street segments. Let  $u$  be the set of unobserved junctions, with no SCATS sensor, and  $-u = V \setminus u$  the junctions with sensors. The saturation of a junction  $v_i$  at a time  $t$  is a continuous random variable  $y_{i,t}$ . Furthermore,  $\mathbf{y}_{u,t} \equiv \{y_{i,t}\}_{i:v_i \in u}$ .

We combine two components to obtain an estimate of the saturation of all junctions at future time steps,  $\mathbf{y}_{V,t+\Delta_t}$ , conditioned on the current observations,  $\mathbf{y}_{-u,t}$  ( $\Delta_t \in \mathbb{N}^0$ ).

The first one,  $P(\mathbf{y}_{-u,t+\Delta_t} | \mathbf{y}_{-u,t})$ , models historical measurements. It can estimate future measurements  $\hat{\mathbf{y}}_{-u,t+\Delta_t}$ , based on the current observations  $\hat{\mathbf{y}}_{-u,t}$ :

$$\hat{\mathbf{y}}_{-u,t+\Delta_t} = E(\mathbf{y}_{-u,t+\Delta_t} | \hat{\mathbf{y}}_{-u,t}) . \quad (1)$$

The second is a Gaussian Process (GP) based on the street network and defining a multivariate Gaussian distribution  $P(\mathbf{y}_{V,t})$  over the saturations at all junctions. Conditioning this distribution on  $\mathbf{y}_{-u}$  provides  $P(\mathbf{y}_{u,t+\Delta_t} | \mathbf{y}_{-u,t+\Delta_t})$  and allows to estimate saturations at junctions without sensors:

$$P(\mathbf{y}_{u,t+\Delta_t} | \mathbf{y}_{-u,t}) \approx P(\mathbf{y}_{u,t+\Delta_t} | \hat{\mathbf{y}}_{-u,t+\Delta_t}) . \quad (2)$$

Figure 1 illustrates the resulting prediction procedure.

## 2. GAUSS-MARKOV

A linear dynamical system models the evolution of a set of state variables  $\mathbf{y} \in \mathbb{R}^p$ , where we omit the subscript  $-u$ :

$$\mathbf{y}_{t+1} = A_t \mathbf{y}_t + \mathbf{w}_t \quad (3)$$

$$\mathbf{w}_t \sim \mathcal{N}(\bar{\mathbf{w}}_t, \Sigma_{w_t}) . \quad (4)$$

$x_1 \sim \mathcal{N}(\bar{\mathbf{y}}_0, \Sigma_0)$ , a multivariate Gaussian distribution of mean  $\bar{\mathbf{y}}_0$  and covariance matrix  $\Sigma_0$ . The Kalman filter can compute  $P(\mathbf{y}_{t+\Delta_t} | \mathbf{y}_t) = \mathcal{N}(\hat{\mathbf{y}}_{t+\Delta_t}, \hat{\Sigma}_{t+\Delta_t})$  recursively.

Sensor measurements were collected from 2013-01-01 to 2013-05-14<sup>1</sup> by 512 (470 non trivial ones) vehicle count sensors located in central Dublin. We average all measurements received on non-overlapping 4 minutes intervals, because of missing values, and model the resulting averages from 5am to 12am. The parameters  $A_t$ ,  $\bar{\mathbf{w}}_t$ ,  $\Sigma_{w_t}$  change for every time step but are identical for every day. So are  $\bar{\mathbf{y}}_0$  and  $\Sigma_0$ .

Following the methodology of [6], each matrix  $A_t$  is learned using (averaged) measurements for  $t' \in \{t - \delta_t, \dots, t + \delta_t\}$ , weighted by a Gaussian kernel:  $\exp(-(t-t')^2/\delta_t)$ . We arbitrarily use  $\delta_t = 3$ . For each matrix  $A_t$ , each row  $\mathbf{r}_{i,t}$  is estimated using an elastic net [7] and ten-fold cross-validation.  $\Sigma_0$  and each  $\Sigma_{w_t}$  are diagonal covariance matrices estimated by maximum likelihood. Alternatively, penalized estimation algorithms such as the graphical lasso [2] could be used.

## 3. GAUSSIAN PROCESS

$P(\mathbf{y}_{u,t+\Delta_t} | \mathbf{y}_{-u,t+\Delta_t})$  is derived from a GP regression framework modeling traffic saturation values of all junctions at a given time, similar to [5]. Multiple sensors at a junction are averaged. For each vertex  $v_i$ , we introduce a latent variable  $f_i$ , the true traffic saturation at  $v_i$ :

$$y_i = f_i + \epsilon_i \quad (5)$$

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2) . \quad (6)$$

We assume that the random vector of all latent variables follows a GP: any finite set  $\mathbf{f} = \{f_i\}_{i=1, \dots, M}$  has a multivariate Gaussian distribution. Therefore, the vector of observed traffic saturations ( $\mathbf{y}_{-u}$ ) and unobserved traffic saturations ( $\mathbf{d}_u$ ) follows a Gaussian distribution

$$\begin{bmatrix} \mathbf{y}_{-u} \\ \mathbf{d}_u \end{bmatrix} \sim \mathcal{N} \left( 0, \begin{bmatrix} K_{-u,-u} + \sigma^2 I & K_{-u,u} \\ K_{u,-u} & K_{u,u} \end{bmatrix} \right), \quad (7)$$

where  $I$  is an identity matrix,  $K$  the so-called kernel and  $K_{u,-u}$ ,  $K_{-u,-u}$ ,  $K_{u,u}$ , and  $K_{-u,u}$  the corresponding entries of  $K$ . Conditioning on  $\mathbf{y}$  produces  $P(\mathbf{y}_{u,t+\Delta_t} | \mathbf{y}_{-u,t+\Delta_t})$ .

We use the common regularized Laplacian kernel function

$$K = [\beta(L + I/\alpha^2)]^{-1}, \quad (8)$$

where  $\alpha$  and  $\beta$  are hyperparameters.  $L$  denotes the combinatorial Laplacian,  $L = D - G$ .  $G$  denotes the adjacency matrix of the graph  $\mathcal{G}$  and  $D$  a diagonal matrix with entries  $d_{i,i} = \sum_j G_{i,j}$ . Variables adjacent in  $\mathcal{G}$  are highly correlated.

## 4. DISCUSSION

We have described a combination of two models able to respectively predict future traffic saturations at junctions with sensors and to extend these predictions to junctions without sensors, in a city. To the best of our knowledge, no similar model has been proposed before.

<sup>1</sup><http://dublinked.ie/datastore/datasets/dataset-305.php>

A similar approach was proposed to provide dynamic cost predictions for a trip planner in the same workshop [4]. Instead of a linear dynamical system (LDS), a spatio-temporal Markov random field (STMRF) is used. It models discretized saturation values only, and inference is approximated by belief propagation whereas it is computationally tractable and performed exactly in LDS. Our model also has a finer temporal resolution. Therefore, it can be used for signaling or online adaptation of the route in addition to offline trip planning. Comparing these two models in terms of precision and speed would be interesting.

The Gauss Markov model assumes the dynamics are linear, first-order Markov and perturbed by Gaussian noise. More refined models could be considered and might lead to better estimations. In particular, we could assume the measurements are noisy observations of a hidden process.

Other information could also be leveraged. For example, the street network could be used to derive a prior on the coefficient of the transition matrix, influencing the model only. Irregular, pointwise traffic estimation (for example based on mobile phones or GPS) could be integrated into the Gaussian Process to produce finer saturation estimates. Finally, different dynamics could be estimated and used in the presence or the absence of rain, modifying both the model and the estimation process.

## 5. ACKNOWLEDGMENTS

This work was supported by the European FP7 project INSIGHT under grant 318225.

## 6. REFERENCES

- [1] A. Artikis, M. Weidlich, F. Schnitzler, I. Boutsis, T. Liebig, N. Piatkowski, C. Bockermann, K. Morik, V. Kalogeraki, J. Marecek, A. Gal, S. Mannor, D. Gunopulos, and D. Kinane. Heterogeneous stream processing and crowdsourcing for urban traffic management. In *Proceedings of the 17th International Conference on Extending Database Technology*, page (to appear), 2014.
- [2] J. Friedman, T. Hastie, and R. Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2008.
- [3] W. Leutzbach. *Introduction to the Theory of Traffic Flow*. Springer, 1988.
- [4] T. Liebig, N. Piatkowski, C. Bockermann, and K. Morik. Predictive trip planning – smart routing in smart cities. In *Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference*, 2014.
- [5] T. Liebig, Z. Xu, M. May, and S. Wrobel. Pedestrian quantity estimation with trajectory patterns. In *Machine Learning and Knowledge Discovery in Databases*, volume 7524 of *Lecture Notes in Computer Science*, pages 629–643. Springer Berlin Heidelberg, 2012.
- [6] L. Song, M. Kolar, and E. P. Xing. Time-varying dynamic bayesian networks. *Advances in Neural Information Processing Systems*, 22:1732–1740, 2009.
- [7] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.



# Sensing Urban Soundscapes

Tae Hong Park<sup>1</sup>, Johnathan Turner<sup>1</sup>, Michael Musick<sup>1</sup>, Jun Hee Lee<sup>1</sup>,  
Christopher Jacoby<sup>1</sup>, Charlie Mydlarz<sup>1,2</sup>, Justin Salamon<sup>1,2</sup>

<sup>1</sup>Music and Audio Research Lab (MARL)  
The Steinhardt School  
New York University  
New York, NY 10012 USA

<sup>2</sup>Center for Urban Science and Progress (CUSP)  
1 MetroTech Center, 19th floor  
New York University  
New York, NY 11201 USA

{thp1, jmt508, musick, junheelee, cbj238, cmydlarz, justin.salamon}@nyu.edu

## ABSTRACT

Noise pollution is one of the most serious quality-of-life issues in urban environments. In New York City (NYC), for example, more than 80% of complaints<sup>1</sup> registered with NYC's 311 phone line<sup>2</sup> are noise complaints. Noise is not just a nuisance to city dwellers as its negative implications go far beyond the issue of quality-of-life; it contributes to cardiovascular disease, cognitive impairment, sleep disturbance, and tinnitus<sup>3</sup>, while also interfering with learning activities [21]. One of the greatest issues in measuring noise lies in two of the core characteristics of acoustic noise itself — transiency and structural multidimensionality. Common noise measurement practices based on average noise levels are severely inadequate in capturing the essence of noise and sound characteristics in general. Noise changes throughout the day, throughout the week, throughout the month, throughout the year, and changes with respect to its frequency characteristics, energy levels, and the context in which it is heard. This paper outlines a collaborative project that addresses critical components for understanding spatio-temporal acoustics: measuring, streaming, archiving, analyzing, and visualizing urban soundscapes [28] with a focus on noise rendered through a cyber-physical sensor network system built on Citygram [23, 24].

## General Terms

Algorithms, Measurement, Design, Reliability, Experimentation, Security, Human Factors, Standardization, Theory.

<sup>1</sup>“Improving Our Quality of Life: Operation Silent Night,” <http://www.nyc.gov>

<sup>2</sup>[http://www.citymayors.com/environment/nyc\\_noise.html](http://www.citymayors.com/environment/nyc_noise.html)

<sup>3</sup>[http://www.euro.who.int/\\_\\_data/assets/pdf\\_file/0008/136466/e94888.pdf](http://www.euro.who.int/__data/assets/pdf_file/0008/136466/e94888.pdf)

## Keywords

Cyber-physical sensor network, data mining, data streaming, big data, acoustics, noise, mobile/distributed computing, mobile data management, soundscapes, machine learning.

## 1. INTRODUCTION

In 1800, a mere 1.7% of the global population lived in cities of 100,000 or more; at the beginning of the 1950s, that number rose to 13% [17], and as of 2013, there are 24 *megacities*<sup>4</sup>, each inhabited by more than 20 million people. By 2050, the projection is that 68% of the global population will dwell in urban areas, which will include more than 37 megacities [31]. The growth of cities has been incredible and when megacities began to emerge in the 1970s (there were only two: NYC and Tokyo [31]), interest in the quality-of-life of city dwellers began to garner the attention of researchers. In the United States, for example, the shift from considering noise as a mere nuisance and an artifact of city-life began to change in the 1970s resulting in Congress putting forth The Noise Pollution and Abatement Act<sup>5</sup>. Research on the impact of noise on urban inhabitants by environmental psychologists also began during this period [6, 5, 10]. Today in NYC, urban noise comprises approximately 80% of all 311 complaints where approximately 40% of the noise complaints are related to loud music, 18% construction noise, and 13% loud talking<sup>6</sup>. With the annual doubling of the world's population, increases in urban noise complaints have the potential to reach alarming levels. However, measuring noise is not trivial. One of the most comprehensive noise codes in the world is The Portland Urban Noise Code<sup>7</sup>, which is based on spatio-temporal metrics of sound. That is, noise is described in terms of location, time, and acoustic energy measurements and is supervised by a control officer with codes enforced by the police department. Noise, however, cannot simply be defined by measuring the decibel (dB) levels at a specific time and place. Furthermore, manually and continuously measuring urban soundscapes is impractical and

<sup>4</sup>Cities that have a population in excess of roughly 20 million people.

<sup>5</sup>42 USC § 4901, 1972

<sup>6</sup><https://nycopendata.socrata.com>

<sup>7</sup>Herman, Paul. Portland, Ord. No. 139931. 1975, amend. 2001. [http://www.nonoise.org/lawlib/cities/portland\\_or](http://www.nonoise.org/lawlib/cities/portland_or)

for all intents and purposes, unfeasible. Our project aims to contribute in developing a comprehensive cyber-physical system to automatically measure, stream, archive, analyze, explore, and visualize acoustic soundscapes with a focus on noise. We begin our paper with a survey of related works, an introduction to the Citygram cyber-physical system and its various modules, and a summary and outline of future work.

## 1.1 Related Work

Quite a few “sound mapping” examples exist including The BBC’s *Save Our Sounds*<sup>8</sup>, *NoiseTube* [18], and *Locustream SoundMap* [16]. Most of the soundmaps are, however, non-real-time and are based on “click and play audio snapshot” interfaces. An example is *Save Our Sounds*, which puts forth the idea of archiving “endangered sounds”. *NoiseTube* and *WideNoise* are two other examples that use crowd-sourcing concepts while utilizing cell-phones’ microphones to measure and share geolocalized dB(A) levels. The *Locustream SoundMap* project is one of the few sound maps that stream real-time audio using an “open mic” concept. In *Locustream*, participants (known as “streamers”) install the developer-provided custom boxes in their apartments and share “non-spectacular or non-event-based quality of the streams.” Other examples include *da\_sense* project [29], which provides a platform for data acquisition, processing, and visualization of urban sensor data (sound, temperature, brightness, and humidity). Their publicly available *NoiseMap* Android application crowd-sources acoustic energy data from participants’ smartphones, which is presented on an online mapping interface with an accompanying public API for data sharing with a daily update rate. Their online platform can also accept data from static sensor networks and has collected over 40,000 data points to date. Commercially available noise monitoring sensors have also been utilized in spatio-temporal urban noise assessments [29]. The discontinued *Tnote Invent* noise-monitoring sensor was used to sample sound levels and transmit them wirelessly at regular intervals. The study focused on the power consumption of these remote sensors, identifying significant power savings when the data transmission strategy is modified, at the cost of increased system latency. A final example is a project called *Sensor City*<sup>9</sup>, which aims to deploy hundreds of static sensing units equipped with acoustic monitoring equipment around a small city in the Netherlands. This solution utilizes its own dedicated fiber-optic network and high-end calibrated audio recording devices. The project is taking a soundscape analysis approach and is looking to investigate human perception and evaluation of acoustic environments within urban settings. The project aims to qualify soundscapes through the development of machine learning algorithms that will analyze incoming data from the sensor network.

## 2. THE CYBER-PHYSICAL SYSTEM

Our interest in this project began in 2011 when observing that current topological mapping paradigms were typically static and focused on visualizing city layouts characterized by slowly changing landmarks such as buildings,

<sup>8</sup><http://www.bbc.co.uk/worldservice/specialreports/saveoursoundsintro.shtml>

<sup>9</sup><http://www.sensorcity.nl>

roads, parking lots, lakes, and other fixed visual objects. Three-dimensional physical shapes, however, do not only define urban environments; they are also defined by “invisible energies” including acoustic energy. Noticing the underrepresentation of sound in modern mapping systems, we began to explore ways to capture spatio-acoustic dimensions and map them on conventional online mapping interfaces. The Citygram project currently involves collaborators from New York University’s (NYU) Steinhardt School, NYU’s Center for Urban Science and Progress (CUSP), and the California Institute of the Arts (CalArts) with support from Google. The project was launched in 2011 to develop methodologies, concepts, and technologies to facilitate the capture and visualization of urban non-ocular energies. The first iteration of Citygram is specifically focused on acoustic data — with such applications as creating dynamic soundmap overlays for online systems, such as Google Maps. Many modern mapping systems have no need to address the issue of temporality. Roads, buildings, and parks do not change on a regular basis; the image update rate for Google Earth, for example, is typically between 1 to 3 years<sup>10</sup>. Such slow update rates, however, are grossly inadequate for mapping sound due to its inherent temporality. The project’s main goal began as an effort to contribute to existing geospatial research by embracing the idea of time-variant, poly-sensory cartography via multi-layered and multi-format data-driven maps based on continuous spatial energies captured by terrestrially deployed remote sensor devices (RSDs). The project’s goals have revolved around creating dynamic, spatio-acoustic maps to help us better understand urban soundscapes and develop technologies to automatically capture man-made, environmental, and machine-made sounds.

NYU CUSP was formed in fall 2013 to utilize “New York City as its laboratory and classroom to help cities around the world become more productive, livable, equitable, and resilient.” CUSP aims to observe, analyze, and model cities “... to optimize outcomes, prototype new solutions, formalize new tools and processes, and develop new expertise/experts.”<sup>11</sup> One of the projects that CUSP has started to focus on is noise in NYC. This has brought together NYU Steinhardt’s Citygram project and CUSP’s initiatives in noise research.

### 2.1 Citygram: System Overview

The Citygram system, which includes three main modules, is shown in Figure 1. The three modules include the RSDs on the left, the server in the middle, and users on the right-hand side. In the following subsections we summarize each module starting with the sensor network and remote sensing devices.

### 2.2 Sensor Network and RSDs

The RSDs form the sensor network which capture, compute, and stream spatio-acoustic data and metadata to the server. The server collects, archives, manages, and analyzes data received from RSDs. We employ two main RSD deployment strategies to create our sensor network as shown in Figure 2: (1) fixed RSDs and (2) crowd-sourced RSDs. Fixed RSDs are installed at fixed locations, which allows

<sup>10</sup><http://sites.google.com/site/earthhowdoi/Home/ageandclarityofimagery>

<sup>11</sup><http://cusp.nyu.edu/about/>

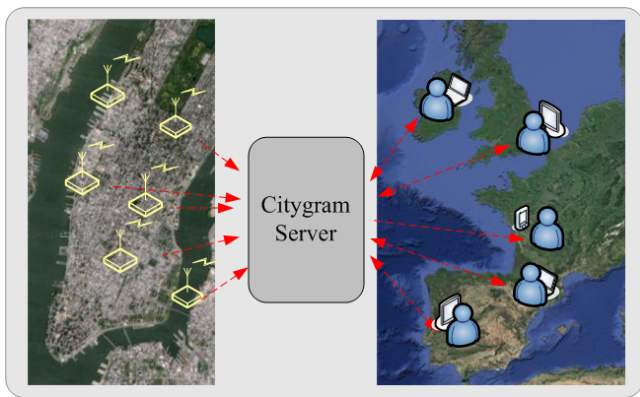


Figure 1: Cyber-physical system.

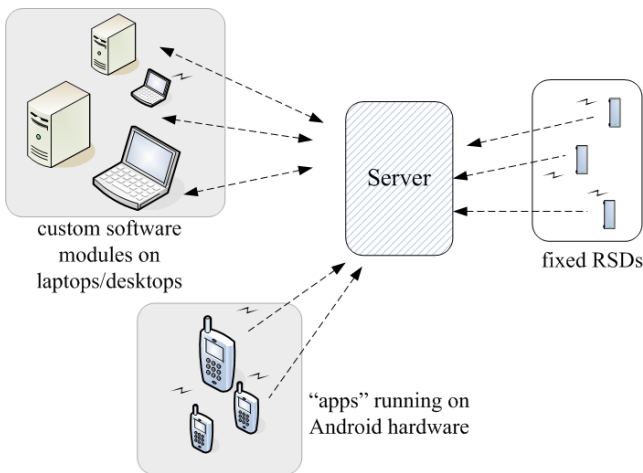


Figure 2: Sensor network showing different RSD types.

for continuous, consistent, and reliable data streams via sensor devices, further detailed below. Difficulties in creating a large-scale, fixed sensor network include selection of appropriate RSDs, as further detailed below, as well as issues concerning deployment in public spaces as discussed in the Future Work section. To create a robust and growing sensor network in lieu of the fixed RSD deployment strategy, we also employ crowd-sourced RSDs to address the issue of coverage expansion, spatial granularity, and engagement of community and citizen scientists. The crowd-sourced RSDs are further divided into mobile apps (e.g. smartphones) and custom software modules for existing personal computer applications that run on software platforms including Max<sup>12</sup> and SuperCollider<sup>13</sup>. The mobile app-based RSDs currently run on the Android OS. Personal computing RSDs only require a microphone and Internet connectivity to stream acoustic data from the user’s device to our server. This crowd-sourced RSD strategy allows for additional measurements that can be especially useful in capturing data that are beyond existing fixed RSD network boundaries.

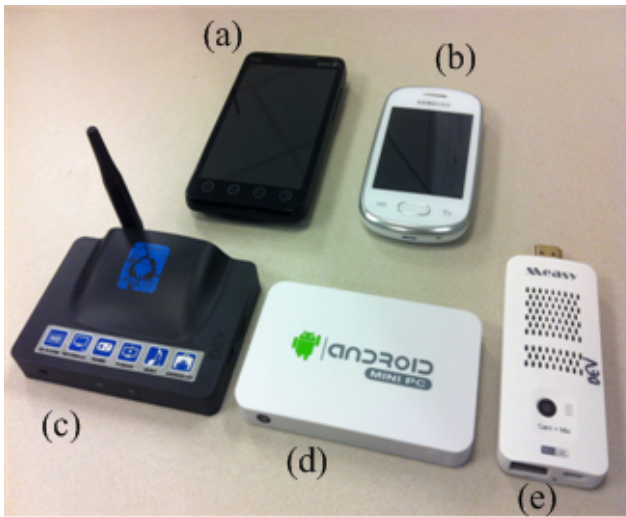
<sup>12</sup><http://cycling74.com>

<sup>13</sup><http://supercollider.sourceforge.net>

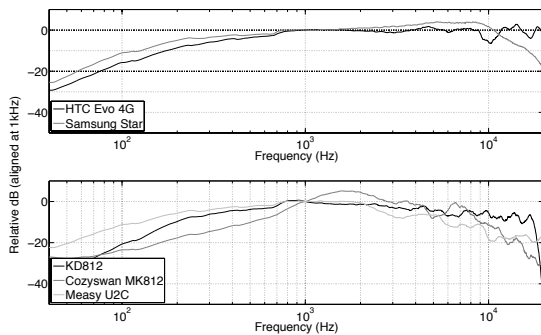
### 2.2.1 Selection of RSDs

A number of hardware platforms have been considered in selecting possible candidates for our fixed RSDs including the Alix system, Raspberry Pi, Arduino, smartphones, and others. Our approach in selecting an appropriate RSD for geospatial acoustic monitoring followed nine criteria: (1) audio capture capability, (2) processing power and RAM, (3) power consumption, (4) flexible OS, (5) onboard storage, (6) wireless connectivity, (7) I/O options/expandability, (8) robustness/sturdiness, and (9) cost. Rather than opting to use custom boards such as the Raspberry Pi, the solution decided upon was Android-based hardware devices including Android mini-PCs. A sub \$50 Android mini-PC, for example, contains the following specifications: quad-core CPU, 2GB RAM, onboard Wi-Fi, Bluetooth, built-in microphone and camera, an HDMI port, and multiple USB ports. A selection of smartphones and mini-PC’s was chosen and run through a preliminary set of objective and subjective tests to determine their suitability for acoustic monitoring in terms of their frequency response and recording quality. The devices are shown in Figure 3, from top left to bottom right: (a) HTC Evo 4G, (b) Samsung Star, (c) KD812, (d) Cozyswan MK812, and (e) Measy U2C. One of the goals of our project is the automatic classification of sound events in urban spaces as further discussed in Section 2.6. To avoid the loss of information, potentially important for the machine learning stage, it is imperative to use appropriate microphones, sampling rates, and bit resolution. During the process of short-listing RSD candidates, we also considered audio capture specifications specifically used in automatic sound classification. The literature seems to suggest varying sampling frequencies. For example, work in classification of soundscape-based sound events in [9] used a 12 kHz sampling rate and in [32] a 16 kHz and 16 bit resolution system was used; in other classification and sound detection work where the focus is on musical signals, filter-banks between 100 Hz–10 kHz [26] and 27.7 Hz–16 kHz were used [3]. Although it is difficult to draw conclusions on the minimal requirements of sample rate and bit resolution parameters, literature in the field of machine learning pertinent to sound seems to suggest that it is not always necessary to have a granularity of 44.1 kHz/16 bit. As part of determining appropriate sound capture parameters, we plan to further investigate the influence of time/frequency resolution on our machine learning algorithms.

To test the potential RSD candidates, environmental recordings were made using all devices simultaneously from a third floor apartment window overlooking a busy urban intersection in Brooklyn, New York. Both of the smartphones sounded clear and individual sound sources were easily identifiable. The frequency bands around the voice range, however, seemed to be accentuated, possibly a result of the devices’ optimization for speech handling. The KD812 and the Cozyswan mini-PCs clearly showed artifacts of dynamic compression. The Measy U2C produced far clearer recordings than the other mini-PCs, with a better low frequency response. Reference sine-sweeps (20 Hz to 20 kHz) were recorded using an Earthworks M30 measurement microphone mounted on-axis placed one meter from a Genelec 8250a loudspeaker in an acoustically soundproofed lab. These first measurements allowed compensation of loudspeaker and room frequency response coloring. The measurement microphone was then replaced by each device with microphone ports on-



**Figure 3: Potential RSDs:** (a) HTC Evo 4G, (b) Samsung Star, (c) KD812, (d) Cozyswan MK812, and (e) Measy U2C.



**Figure 4: Frequency response of Android devices.**

axis to the loudspeaker. The same sine-sweeps were then repeated for each device. Impulse responses were extracted from each sweep using deconvolution techniques, generating device frequency responses as shown in Figure 4. Each device response was level-normalized at 1 kHz.

All devices show a relatively poor response at lower frequencies. This is especially the case for Cozyswan MK812 mini-PC. The large peak in the device’s frequency response at 1–3 kHz highlights the perceived coloration of the environmental recording. The enhanced high frequency response from the HTC and Samsung smartphone devices can also be observed. Between 2–10 kHz the response varies between device types with the most amount of high frequency drop-off occurring on the mini-PC’s. All of the microphones on the mini-PC’s were mounted on the internal printed circuit board facing up, explaining the filtering effects caused by the small microphone port coupled to the relatively large internal cavity of the device. Based on our tests, the smartphones seemed to provide the preferred solution. However, to match the processing power of the mini-PC’s, a high-end smartphone would be required. Furthermore, audio recording quality of the Measy U2C’s built-in microphone is com-

parable to the tested smartphones with the added benefit of increased processing power and expandability. This expandability allows for the connection of external USB microphones, providing the potential for consistent and enhanced audio quality as well as flexibility in microphone placement. The external USB microphone allows the device itself to be kept modular and separate from the mini-PC, allowing flexibility in its placement and replacement. We are currently in the process of investigating the use of external USB audio peripherals.

### 2.3 Signal Processing and Feature Extraction

The RSDs autonomously capture audio in real-time and employ distributed computing strategies to address efficiency and scalability issues by running various computational tasks including feature extraction. This strategy alleviates stress on the server in the context of sensor network scalability and efficiency. Blurred acoustic data and non-invertible low-level acoustic features are streamed to the server.

Our current custom Android software, which runs both on our fixed sensor network as well as on conventional Android-based smartphones, uses OpenSL ES<sup>14</sup> to record audio blocks using circular buffering techniques. As shown in Figure 5, feature vectors are transmitted as JSON<sup>15</sup> objects via HTTP posts. The raw audio signal is compressed and streamed to the server where users can then monitor soundscapes in real-time. Before transmission to our server, the raw audio is passed through a speech-blurring algorithm as further detailed below. For development of machine learning algorithms we also have the option to stream raw audio data using libsndfile<sup>16</sup> with FLAC codec<sup>17</sup>, which is separately archived on the server and inaccessible by the public.

To enable users to hear the “texture” and characteristics of spaces without compromising the privacy of people in the monitored areas, we employ a modified granular synthesis technique [27] to blur the spectrum of the voice. This is achieved prior to streaming the raw audio data to our server where users can access the blurred audio streams. To accomplish these conflicting tasks — blurring the audio while retaining the soundscape’s texture — a multi-band signal processing approach was devised. The audio signal is first divided into three sub-bands using overlapping filter windows with cascaded third-order Butterworth filters. The middle band represents the voice band. The isolated voice band is then segmented into 32 windows, or “grains”, with 50% window overlap. Blurring is achieved by randomizing the temporal order of the windows. The randomized grains are then used to construct the blurred voice band signal via overlap-and-add [4, 25]. The final audio signal sent to the server is a sum of the unaltered lower/upper bands and the modulated voice band. Not shown in Figure 5 is a separate audio thread and audio streaming block that transmits raw audio data for machine learning research purposes. This module will eventually be removed from our system once our algorithms are fully developed.

### 2.4 Server and Database

The basic structure of the Citygram system is comprised of users, RSDs, and the Citygram server. A user may reg-

<sup>14</sup><http://www.khronos.org/opensles>

<sup>15</sup><http://www.json.org>

<sup>16</sup><http://www.mega-nerd.com/libsndfile>

<sup>17</sup><http://xiph.org/flac>



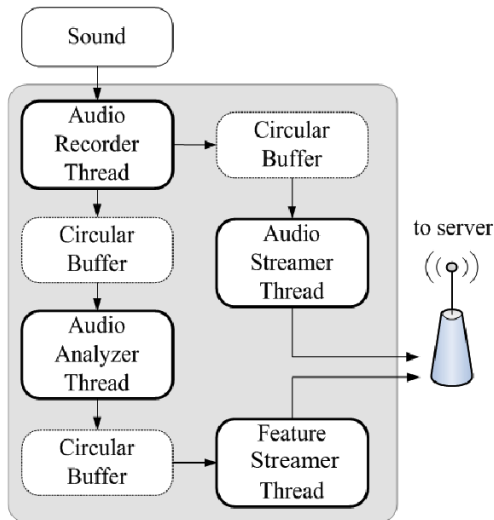


Figure 5: Block diagram of the Android software.

ister multiple RSDs enabling simultaneous streaming from multiple locations. The server currently uses MySQL to store all data and metadata. The information submitted by each RSD is stored in our database where feature values and references are stored in tables. Feature vectors include time- and frequency-domain data, a compressed representation of the audio’s discrete Fourier transform, and UTC<sup>18</sup> timestamps. For timestamp synchronization across platforms, each RSD uses OS-specific timing libraries that provide microsecond granularity; a server-side cron job regularly updates and synchronizes its internal clock to network UTC time. Metadata information for all the RSDs is also stored in tables and includes RSD latitude and longitude drawn from the Google Maps Geolocation API<sup>19</sup>, and the device’s activity status. Additionally, the Citygram website holds a separate database of all user data, including number of RSDs and platform information. Feature pushing is accomplished through simple HTTP POST requests using the cURL library<sup>20</sup> and pulling is accomplished using a GET request. All requests are formatted using JSON. This relatively simple mechanism allows for consistency across all platforms.

## 2.5 User Interaction and Visualization

The entry point for becoming a streamer is the Citygram website<sup>21</sup>. The website is the central hub for management and communication between devices and users. A user registers through the site and creates a username and password. This registration information, along with details involving location and platform type, are stored in the database before being linked to specific RSDs. Within this portal, a user may edit the location of their RSDs as well as view an interactive visualization of the currently streaming devices. Beyond administrative roles, two types of users exist on the Citygram site: default user and researcher. A default user

<sup>18</sup><http://www.time.gov>

<sup>19</sup><https://developers.google.com/maps/documentation/business/geolocation>

<sup>20</sup><http://curl.haxx.se>

<sup>21</sup><http://citygram.smusic.nyu.edu>



Figure 6: Dynamic soundmap snapshot.

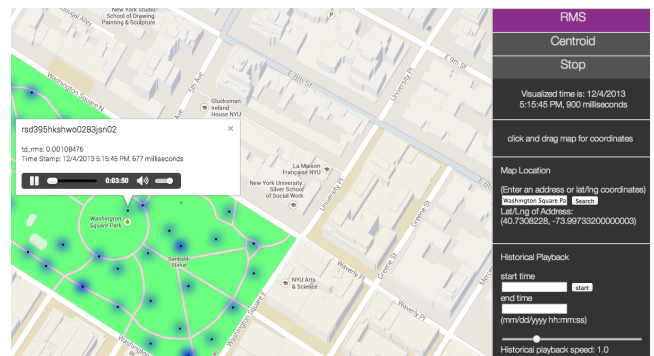


Figure 7: Screenshot of interface.

may register up to 50 RSDs and is restricted in their access to historical data. In contrast, a “researcher user” has no cap on the amount of devices that can be registered and may access all historical data. Additionally, researchers may also register other users.

In order for Citygram to attract potential streamers, users, urban scientists, the general public, and artists, we have developed a number of interfaces and software applications. This includes tools for commonly used software environments such as MATLAB, Max, Processing, and SuperCollider. Once registered through the Citygram website, a user may easily enter their username and device ID during the initialization of a streaming session. This will allow a number of different interaction modes with our server including pulling data from select RSDs.

The Citygram system currently provides quasi-real-time (subject to network and buffering latency) visualizations via standard web browsers such as Google Chrome. The interface dynamically visualizes RSD-streamed audio features and also provides the ability to visualize historical data stored in the database.

The web interface is designed to function as a spatio-acoustic exploration portal. By default, data from all RSDs on the map are visualized. However, it is also possible for a user to select a specific area on the map to visualize only a subset of RSDs. Our maps are built on the Google Maps API. Each RSD is represented by a marker, which is user-selectable for additional information and extended function-

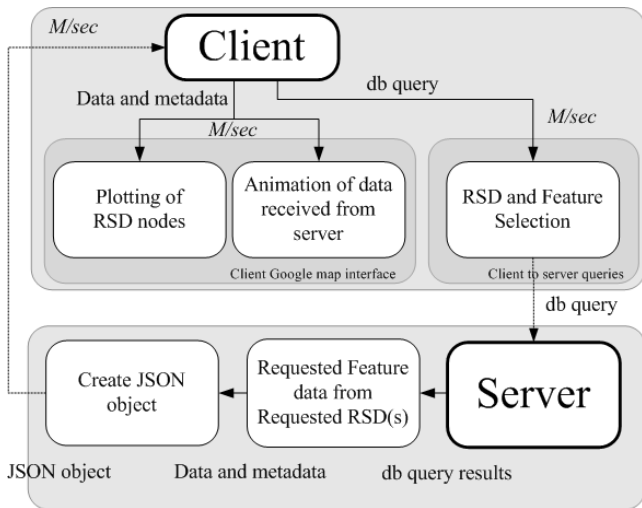


Figure 8: Data animation in Citygram.

ality. Clicking on the marker brings up a window populated by the latitude/longitude coordinates and address, real-time feature vector values, photo snapshot from Google Street View, the precise feature timestamp, and an audio monitoring button that allows the user to hear RSD-specific real-time audio streams. Our current visualizations utilize heatmaps to dynamically display low-level acoustic feature data streamed by RSDs as shown in Figure 6.

Additional features include controls that can be used to enable/disable feature visualizations. Once a user selects features for visualization, the client computer queries the server for its current timestamp and synchronizes its animation to this timestamp. This timestamp is used to query the list of available RSDs. A two second buffer is created, containing data from RSDs that are active for the specified time duration. This list is returned to the client, which currently animates the 2-seconds worth of samples at a rate of 10 frames per second (cf. Figure 6). These parameters are adjustable. If an RSD streams at a lower rate than the visualizer’s frame rate, frame compensation takes place: a sample-and-hold technique is employed to previous samples, which are held across animation frames. Additionally, the control menu has an optional section in which to enter a start and end time for historical animation. The playback speed of historical data is controllable by a slider facilitating browsing of past data sets.

## 2.6 Analytics

One of the key goals of the project is to automatically analyze and classify audio data captured by our sensor network. This includes identifying urban sounds such as sirens, car horns, street music, children playing, dogs barking, wind blowing, and automobiles humming and idling. In the context of measuring and identifying sounds that can be considered noise, the initial stages of our research is framed within soundscape analysis and auditory scene analysis [7, 14, 28, 34]. This allows us to go beyond the somewhat simplistic “more decibels equal more noise” paradigm. Soundscape research typically begins with source identification [8], and as such, we believe automatic source identification to be a key research component for our project. Automatic source iden-

tification can be further divided into acoustic event detection (AED) and acoustic event classification (AEC) where the former refers to providing a semantic label [1, 13, 12] and the latter assigning temporal level segmentation that can then be used for AEC [1, 9, 12, 19].

In order to develop supervised learning systems, we require large annotated datasets. Annotated datasets for music, speech, and birds are readily available. For example, for music there exists the Million Song Dataset [2], CAL500 (500 songs with multiple annotations) [33], Last.fm (960,000 tags) [15], McGill Billboard dataset<sup>22</sup>; bird sound examples include HU-ASA [11], Cornell-Macaulay Library of Natural Sounds, Peterson Field Guides, Common Bird Songs, and Mirtovic’s database. For general acoustic events (especially outdoor spaces) databases are currently scarce. Only a few exist and include the CLEAR<sup>23</sup> and C4DM D-CASE dataset [12]. Both datasets, however, only contain annotated sound samples primarily recorded within office spaces. We are, therefore, in the process of collecting and annotating data. One of the data sources we are using in the meantime is Freesound<sup>24</sup> which provides an API for accessing crowd-sourced and annotated audio samples that are keyword searchable (e.g. city + noise). There are a number of issues with Freesound as a source for ground-truth data including singular annotations, varying audio quality, and potentially erroneous or irrelevant labeling. In parallel to collecting and creating annotated datasets, we are also working on the development of an urban sound taxonomy to determine classes of sound sources we are interested in having our AED/AEC system identify. The taxonomy efforts will also serve in contributing towards creating a coherent framework for dealing with sounds both within the project and when relating to the existing literature on soundscape research.

## 3. SUMMARY

NYC is one of the largest, busiest, and most complex cities in the world. In the past two years, the city has on average received 227 noise complaint calls per day, made by city dwellers utilizing NYC’s 311 hotline. In the years to come, these statistics are projected to get worse as city population growth is expected to increase significantly on a global scale. The current state of noise measurement and control infrastructures based on average dB levels are ineffective in capturing the spectral, temporal, and spatial characteristics of noise. Our early efforts and interest in cyber-physical systems to address this issue began with research and development of dynamic cartographic mapping systems to explore non-ocular urban energies. This resulted in creation of the Citygram project in 2011. In its first iteration, Citygram’s focus was aimed towards acoustic energy — i.e. soundscapes that included all types of sound. More recently, in collaboration with NYU CUSP, however, we have narrowed our scope of research to spatio-acoustic noise. In this paper, we have outlined our cyber-physical system that addresses the acquisition, archival, analysis, and visualization of sound captured from urban spaces. The various components discussed in our paper included cross-platform remote sensing devices (RSDs), data acquisition and crowd-sourcing strategies for

<sup>22</sup><http://ddmal.music.mcgill.ca/billboard>

<sup>23</sup><http://www.clear-evaluation.org>

<sup>24</sup><http://www.freesound.org>

data streaming, sensor network designs, summarizing our database architecture, our Internet exploration portal, and work concerning analytics. We expect that our research and development outcomes will contribute towards creating multimodal interactive digital maps based on poly-sensory RSDs to quantitatively measure and represent soundscapes and acoustic noise in real-time; provide interactive spatio-acoustic software tools for researchers, educators, the general public, and citizen scientists; and also contribute to urban planning, noise city code development, and improving the quality-of-life of city inhabitants.

#### 4. FUTURE WORK

To address the complexities related to sensor network deployment and scalability, the next stage of the project will be focused on small-scale RSD deployment. Two NYC parks have been chosen for deployment of 40 RSDs. These two sites will allow the project infrastructure to be rigorously tested under outdoor weather conditions, which will help us gain valuable insights into Wi-Fi connectivity, equipment malfunction/damage, system performance under various weather conditions, and power supply issues. We also plan to set up another sensor network of 30 RSDs on the CalArts campus near Los Angeles. Our RSDs will stream in-situ raw indoor and outdoor soundscape data to our server in an effort to create a large ground truth dataset: this dataset will be annotated and labeled.

Permanently fixed mounting locations are also presently being considered. Ongoing efforts include working with cities to deploy RSDs through existing infrastructures that already include communication and power supply solutions. One such infrastructure is the urban payphone system, which will become, or already is, functionally irrelevant. NYC, for example, is planning to repurpose the payphone system: of the currently active 12,360 public payphones, 10 have recently been converted to include free Wi-Fi, with more planned for the future. This infrastructure is ideal for our project, as it would quickly lead to the availability of a large number of nodes throughout the city employing what we call “mount and play” strategy — mounting inexpensive, highly sophisticated, and robust RSDs onto existing urban infrastructures such as payphones, electronic parking meters, and traffic control systems.

The acoustic data obtained through our sensor network will also be used to investigate connections between spatio-acoustic characteristics and existing geolocated datasets, such as crime statistics, weather patterns, school attainment metrics, municipal/census data and public social network feeds, and real-estate statistics which can provide rich quantitative and contextual location-based information. Another type of information that we are interested in is environmental emotion/mood. Although this interdisciplinary research is still in its nascent stages, automatic mood detection has found increasing interest in the field of music, speech analysis, face-recognition, and natural language processing [22, 20, 30, 35]. Much of the emotion/mood detection research for sound has been in the realm of music. However, there is strong potential that algorithms and methodologies used in music will translate to urban acoustic signals as: (1) music is omnipresent in urban spaces and (2) many of the low-level feature vectors are timbral rather than musical and reflect acoustic dimensions of sound. Voice blurring will be another area for further research and development. To im-

prove sound monitoring quality without compromising private speech that might be captured by the RSDs, voice activity detection (VAD) techniques will be explored. Currently, voice blurring occurs regardless of the absence of speech in the soundscape. With the inclusion of VAD, blurring will only occur when speech is detected by an RSD.

Another key goal for the project is developing effective and informative visualizations by embracing the idea that “a [moving] picture is worth a thousand [trillions of] words.” In order to reach our visualization goals, we will: (1) develop robust and accurate sound classification algorithms, (2) design interactive visualizations to effectively present an ocean of data that is continuously in flux, and (3) integrate other spatial data and provide unified multi-data visualizations.

Today’s megacities we inhabit are very complex. Future megacities and their soundscapes will become even more complex and will likely dwarf the noise and loudness levels of today’s cities. To improve the quality-of-life of current and future city-dwellers, we need to better understand urban spaces. However, “you can’t fix what you can’t measure.” Therefore, our hope is to contribute towards developing a cyber-physical system to dynamically and quantitatively measure and “sense” urban soundscapes and ultimately help improve the quality-of-life of city communities.

#### 5. ACKNOWLEDGMENTS

Our thanks to Google and NYU CUSP for their support.

#### 6. REFERENCES

- [1] J.-J. Aucouturier, B. Defreuil, and F. Pachet. The bag-of-frames approach to audio pattern recognition: a sufficient model for urban soundscapes but not for polyphonic music. *The Journal of the Acoustical Society of America*, 122(2):881–91, Aug. 2007.
- [2] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24–28, 2011, Miami, Florida*, pages 591–596. University of Miami, 2011.
- [3] S. Böck, F. Krebs, and M. Schedl. Evaluating the Online Capabilities of Onset Detection Methods. In *ISMIR*, pages 49–54, 2012.
- [4] M. Bradshaw and I. Xenakis. Formalized Music: Thought and Mathematics in Composition. *Music Educators Journal*, 59(8):85, Apr. 1973.
- [5] A. L. Bronzaft. The effect of a noise abatement program on reading ability. *Journal of environmental psychology*, 1(3):215–222, 1981.
- [6] a. L. Bronzaft and D. P. McCarthy. The Effect of Elevated Train Noise On Reading Ability. *Environment and Behavior*, 7(4):517–528, 1975.
- [7] A. L. Brown, J. Kang, and T. Gjestland. Towards standardization in soundscape preference assessment. *Applied Acoustics*, 72(6):387–392, 2011.
- [8] L. D. Brown, H. Hua, and C. Gao. A widget framework for augmented interaction in SCAPE. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 1–10. ACM, 2003.
- [9] C. V. Cotton and D. P. W. Ellis. Spectral vs. spectro-temporal features for acoustic event detection.



- In *Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2011 IEEE Workshop on, pages 69–72. IEEE, 2011.
- [10] G. W. Evans and S. J. Lepore. Nonauditory effects of noise on children: A critical review. *Children's environments*, pages 31–51, 1993.
- [11] K.-H. Frommolt, R. Bardeli, F. Kurth, and M. Clausen. *The animal sound archive at the Humboldt-University of Berlin: Current activities in conservation and improving access for bioacoustic research*. Slovenska akademija znanosti in umetnosti, 2006.
- [12] D. Giannoulis, E. Benetos, D. Stowell, M. Rossignol, M. Lagrange, and M. Plumbley. IEEE AASP challenge: Detection and classification of acoustic scenes and events. Technical report, Technical Report, Queen Mary University of London, 2013.
- [13] D. Giannoulis, D. Stowell, E. Benetos, M. Rossignol, M. Lagrange, and M. D. Plumbley. A database and challenge for acoustic scene classification and event detection. *submitted to Proc. EUSIPCO*, 2013.
- [14] C. Guastavino. Categorization of environmental sounds. *Canadian journal of experimental psychology = Revue canadienne de psychologie expérimentale*, 61(1):54–63, Mar. 2007.
- [15] V. Henning and J. Reichelt. Mendeley-A Last. fm For Research? In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 327–328. IEEE, 2008.
- [16] J. Joy and P. Sinclair. Networked music & soundart timeline (NMSAT): a panoramic view of practices and techniques related to sound transmission and distance listening. *Contemporary Music Review*, 28(4-5):351–361, 2009.
- [17] J. J. Macionis and V. N. Parrillo. *Cities and urban life*. Pearson Education, 2004.
- [18] N. Maisonneuve, M. Stevens, and M. E. Niessen. NoiseTube: Measuring and mapping noise pollution with mobile phones. *Environmental Engineering*, (May), 2009.
- [19] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen. Acoustic event detection in real life recordings. In *18th European Signal Processing Conference*, pages 1267–1271, 2010.
- [20] O. C. Meyers. *A mood-based music classification and exploration system*. PhD thesis, Massachusetts Institute of Technology, 2007.
- [21] A. Nadakavukaren. *Our global environment: A health perspective*. Waveland Press Prospect Heights, IL, 2000.
- [22] R. Nagpal, P. Nagpal, and S. Kaur. Hybrid Technique for Human Face Emotion Detection. *International Journal on Advances in Soft Computing and Its Applications (IJASCA)*, 1(6):87–90, 2010.
- [23] T. H. Park, B. Miller, A. Shrestha, S. Lee, and J. Turner. Citygram One: Visualizing Urban Acoustic Ecology. *Digital Humanities 2012*, 13(6):313, 2012.
- [24] T. H. Park, J. Turner, C. Jacoby, A. Marse, M. Musick, A. Kapur, and J. He. Locative Sonification: Playing the World Through Citygram. ICMC, 2013.
- [25] L. R. Rabiner and B. Gold. Theory and application of digital signal processing. *Englewood Cliffs, NJ, Prentice-Hall, Inc.*, 1975. 777 p., 1, 1975.
- [26] J. Ricard. An implementation of multi-band onset detection. *integration*, 1(2):10, 2005.
- [27] C. Roads. Introduction to granular synthesis. *Computer Music Journal*, 12(2):11–13, 1988.
- [28] R. M. Schafer. *The tuning of the world*. Knopf, 1977.
- [29] I. Schweizer, R. Bärtl, A. Schulz, F. Probst, and M. Mühläuser. NoiseMap-real-time participatory noise maps. In *Proc. 2nd Int'l Workshop on Sensing Applications on Mobile Phones (PhoneSense'11)*, pages 1–5, 2011.
- [30] I. Shafran and M. Mohri. A comparison of classifiers for detecting emotion from speech. In *Proc. ICASSP*, 2005.
- [31] M. Sivak and S. Bao. Road safety in New York and Los Angeles: US megacities compared with the nation. 2012.
- [32] A. Temko, R. Malkin, C. Zieger, D. Macho, C. Nadeu, and M. Omologo. Acoustic event detection and classification in smart-room environments: Evaluation of CHIL project systems. *Cough*, 65(48):5, 2006.
- [33] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Towards musical query-by-semantic-description using the cal500 data set. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 439–446. ACM, 2007.
- [34] D. Wang, G. J. Brown, and Others. *Computational auditory scene analysis: Principles, algorithms, and applications*, volume 147. Wiley interscience, 2006.
- [35] Y. Xia, L. Wang, and K.-F. Wong. Sentiment vector space model for lyric-based song sentiment classification. *International Journal of Computer Processing Of Languages*, 21(04):309–330, 2008.

# Privacy and Anonymity in the Information Society (PAIS)

Traian Marius Truta  
Li Xiong  
Farshad Fotouhi

# A Hybrid Approach for Privacy-preserving Record Linkage

Murat Kantarcioglu  
Department of Computer Science  
The University of Texas at Dallas  
Richardson, TX 75080, USA  
muratk@utdallas.edu

## ABSTRACT

The integration of information dispersed among multiple repositories is a crucial step for accurate data analysis in various domains. In support of this goal, it is critical to devise procedures for identifying similar records across distinct data sources. At the same time, to adhere to privacy regulations and policies, such procedures should protect the confidentiality of the individuals to whom the information corresponds. Various private record linkage (PRL) protocols have been proposed to achieve this goal, involving secure multi-party computation (SMC) and similarity preserving data transformation techniques. SMC methods provide secure and accurate solutions to the PRL problem, but are prohibitively expensive in practice for large data sets, mainly due to excessive computational requirements. Data transformation techniques offer more practical solutions, but incur the cost of information leakage and false matches.

In this talk, we discuss how the performance of SMC based PRL techniques could be significantly improved by combining them with data sanitization techniques without incurring the cost of information leakage and false matches. Furthermore, we discuss how to efficiently handle typographical errors exist in data during the PRL protocol execution.

## BIO

Dr. Murat Kantarcioglu is an Associate Professor in the Computer Science Department and Director of the UTD Data Security and Privacy Lab at the University of Texas at Dallas. He holds a B.S. in Computer Engineering from Middle East Technical University, and M.S. and Ph.D degrees in Computer Science from Purdue University. He is a recipient of NSF CAREER award and Purdue CERIAS Diamond Award for Academic excellence. Currently, he is a visiting scholar at Harvard Data Privacy Lab.

Dr. Kantarcioglu's research focuses on creating technologies that can efficiently extract useful information from any data without sacrificing privacy or security. His research has been supported by grants from NSF, AFOSR, ONR, NSA, and NIH. He has published over 100 peer reviewed papers. Some of his research work has been covered by the media outlets such as Boston Globe, ABC News etc. and has received two best paper awards.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

*7th International Workshop on Privacy and Anonymity in the Information Society (PAIS'14)* March 28, 2014, Athens, Greece

# Clustering-based Multidimensional Sequence Data Anonymization

Morvarid Sehatkar  
University of Ottawa  
Ottawa, ON, Canada  
msehatkar@uottawa.ca

Stan Matwin  
<sup>1</sup>Dalhousie University  
Halifax, NS, Canada  
<sup>2</sup>Institute for Computer Science of the  
Polish Academy of Science  
Warsaw, Poland  
stan@cs.dal.ca

## ABSTRACT

Sequence data mining has many interesting applications in a large number of domains including finance, medicine, and business. However, Sequence data often contains sensitive information about individuals and improper release and usage of this data may lead to privacy violation. In this paper, we study the privacy issues in publishing multidimensional sequence data. We propose an anonymization algorithm, using hierarchical clustering and sequence alignment techniques, which is capable of preventing both identity disclosure and sensitive information inference. The empirical results show that our approach can effectively preserve data utility as much as possible, while preserving privacy.

## Categories and Subject Descriptors

H.2.7 [Database Management]: Database Administration-- Security, integrity, and protection

## General Terms

Algorithms, Performance, Experimentation, Security

## Keywords

Data anonymization, privacy, multidimensional sequence data, longitudinal data, clustering,  $k$ -anonymity

## 1. INTRODUCTION

Recent advances in information technology have enabled public organizations and corporations to collect and store huge amounts of individuals' data in data repositories. Such data are powerful sources of information about an individual's life such as interests, activities, and finances. Corporations can employ data mining techniques to extract useful knowledge from individuals' data and exploit this knowledge to improve their strategic decision making, enhance business performance, and improve services. As a result, the demand for collecting and sharing data has been rapidly increased. Among various types of individuals' data, event sequence data mining has many interesting applications in a large number of domains. Sequence data mining enables us to discover behaviour patterns of individuals through temporal activities. Such knowledge is precious for planning, detecting behavioral changes, and commercial purposes. For instance, longitudinal medical records of patients can be used to analyze patients' reactions to a new drug or to support a diagnosis. However, despite all benefits of analyzing event sequence data, this data often contain sensitive information and may violate privacy of

individuals if published. In event sequence data, every event may have a number of attributes that act as *quasi-identifiers* ( $QIs$ ). Due to temporal correlation among the events of each sequence, in addition to the values of  $QIs$  within an event, any combination of  $QIs$  values across events along with the temporal information about these values might lead to privacy breach. For example, consider Table 1 containing information of multiple visits of patients in a hospital over the last five years. Every visit corresponds to a multidimensional event and the ordered list of these events represents one sequence. Each event has 5 attributes, including admission year ( $AdmYr$ ), ZIP code, number of days since the first visit in each year ( $DSFC$ ), and the length of stay in the hospital ( $LOS$ ), which all act as  $QIs$ , as well as one sensitive attribute diagnosis. An adversary with some background knowledge about visits of a target individual is able to launch two types of privacy attacks: *identity disclosure* and *attribute disclosure*. For instance, if the adversary knows that Bob had a visit in 2009 and he has been living in ZIP code 56230 from 2010, she can uniquely identify Bob's record, #6, and consequently conclude that Bob has *HIV*. In case of attribute disclosure, if the adversary knows that Bob had a visit in 2007 and *later* in 2011 he was hospitalized for 3 days, then she can conclude that Bob has *HIV* since both matching records to her knowledge, #8 and #9, have *HIV* in one of their visits.

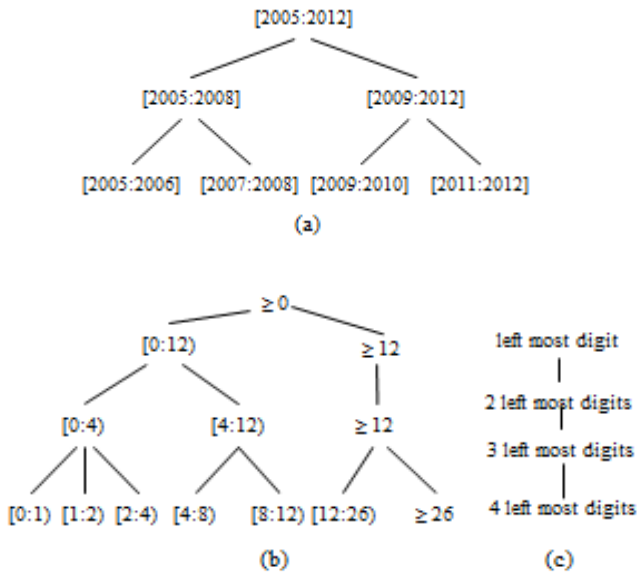
A common practice for releasing individuals' data without violating privacy is *data anonymization*. Data anonymization techniques aim to modify data such that no sensitive information about individuals can be disclosed from published data while data distortion is minimized to ensure usefulness of data in practice. In order to effectively anonymize multidimensional sequence data, to prevent both identity disclosure and attribute disclosure attacks, temporal correlation among the events of each record should be considered in anonymization process, and it should be guaranteed that no combination of values of  $QIs$  within an event and across events of any record leads to privacy breach. In the past years, several anonymization algorithms were proposed to protect privacy when publishing different types of data [2]. However, none of these methods are applicable to anonymize a multidimensional sequence dataset, like the data in Figure 1 (a). Recently, a few methods have been designed to anonymize longitudinal health data which is a case of event sequence data [1][6][7]. However, authors in [1] and [7] only focused on privacy protection against identity disclosure. Moreover, in the longitudinal data, studied in [7], each record contains a sequence of ( $ICD$ ,  $Age$ ) pairs as well as a DNA sequence where  $ICD$  represents the code of the diagnosis made for a patient and  $Age$  is the patient's age at the time of diagnosis. Considering such data, background knowledge of an adversary in this method is modeled as any combination of ( $ICD$ ,  $Age$ ) pairs. Obviously, this method

**Table 1 Patient data of a hospital**

PID	VID	AdmYr	ZIP	DSFC	LOS	Disease
1	1	2009	56117	0	3	Hepatitis
2	1	2007	56103	0	2	Infection
3	1	2008	56942	0	1	Fever
3	2	2010	56942	0	30	Infection
4	1	2008	56107	0	2	Fever
4	2	2010	56107	0	35	Flu
5	1	2009	56117	0	3	Fever
6	1	2009	56103	0	3	Flu
6	2	2009	56103	10	1	Fever
6	3	2010	56230	0	2	HIV
7	1	2008	56072	0	2	Flu
8	1	2007	56361	0	30	Hepatitis
8	2	2011	56107	0	3	HIV
9	1	2007	56230	0	35	Flu
9	2	2011	56107	0	3	HIV
10	1	2009	56072	0	2	Flu
10	2	2009	56103	13	35	Fever
10	3	2010	56043	0	30	Infection

**Table 2 Anonymized patient data satisfying (2, 0.5)-privacy**

PID	VID	AdmYr	ZIP	DSFC	LOS	Disease
1	1	2009	56117	0	3	Hepatitis
2	1	[2007:2008]	56***	0	2	Infection
3	1	[2007:2008]	56***	0	[0:12]	Fever
3	2	[2009:2012]	56***	0	[0:12]	Infection
4	1	[2007:2008]	56***	0	[0:12]	Fever
4	2	[2009:2012]	56107	0	[0:12]	Flu
5	1	2009	56117	0	3	Fever
6	1	2009	56***	0	[0:1]	Flu
6	2	2009	56103	[1:2]	[0:12]	Fever
6	3	2010	56***	0	[0:12]	HIV
7	1	[2007:2008]	56***	0	2	Flu
8	1	[2007:2008]	56***	0	[0:12]	Hepatitis
8	2	[2009:2012]	56107	0	[0:12]	HIV
9	1	[2007:2008]	56***	0	[0:12]	Flu
9	2	[2009:2012]	56***	0	[0:12]	HIV
10	1	2009	56***	0	[0:1]	Flu
10	2	2009	56103	[1:2]	[0:12]	Fever
10	3	2010	56***	0	[0:12]	Infection



**Figure 1 Generalization hierarchy for (a) AdmYr (b) DSFC and LOS in terms of number of weeks (c) ZIP**

is limited to two  $QIs$  and fails to consider the multidimensionality of events in our problem. In [1] it is assumed that adversaries would not have any information about co-occurrence of values of quasi-identifiers in one event as well as the order of events of a target individual. As a result this work fails to model all potential background knowledge of adversaries. The proposed method in [6] prevents both identity disclosure and attributes disclosure; however knowledge of adversaries is assumed to be limited to at most  $p$  values of quasi-identifiers. Although this assumption decreases information loss, determining the appropriate value of  $p$  is not trivial. As a result the adequate level of privacy protection may not be achieved. In this paper, we define a new privacy model called  $(k,c)$ -privacy to anonymize multidimensional sequence data to prevent identity disclosure and attribute disclosure. This privacy model ensures that every combination of values of  $QIs$  within an event and across events of any sequence is shared by at least  $k$  sequences, and the probability of inferring any sensitive value is at most  $c$ . We achieve  $(k, c)$ -privacy by presenting an anonymization algorithm based on *hierarchical agglomerative clustering* [4] and *sequence alignment* [5] techniques. We assume that the purpose of data publication is unknown and so our algorithm anonymizes data by minimizing overall data distortion. Table 2 shows an anonymized version of the data in Figure 1(a) satisfying  $(2, 0.5)$ -privacy using generalization hierarchies in Figure 1.

## 2. PROBLEM DEFINITION

In this section we present the framework which forms the basis of our anonymization methodology. Specifically, we describe the privacy model and the utility measure.

### 2.1 Privacy Model

Suppose a data holder wants to share its multidimensional sequence data for public use. Let  $A = \{A_1, A_2, \dots, A_n\}$  be a set of attributes and  $\Delta = \{\Delta_1, \Delta_2, \dots, \Delta_n\}$  be the corresponding attribute domains. Each  $A_z$  is either a categorical or a numerical attribute. Also assume there is one sensitive attribute  $\psi$  with the domain values  $\Delta_\psi = \{s_1, \dots, s_l\}$ . A multidimensional sequence dataset  $D$  is a collection of records of the form  $(SID, S)$ , where  $SID$  is a unique id for every individual and  $S$  is an ordered list of multidimensional events, denoted by  $S = \langle e_1, e_2, \dots, e_m \rangle$ . Each event  $e$  has the form  $(EID, a_1, a_2, \dots, a_n, s)$  where  $EID$  is the event's id,  $a_z$  is a domain value of  $A_z$ ,  $a_z \in \Delta_z$ , and  $s$  is a value of the sensitive attribute  $\psi$ ,  $s \in \Delta_\psi$ . Events of every sequence  $S$  are ordered with respect to temporal information of one of the attributes  $A_z \in \{A_1, A_2, \dots, A_n\}$ . We refer to the value of the  $z^{th}$   $QI$  attribute of the  $j^{th}$  event of the sequence  $S_i$  by  $e_{ij}(z)$  and the value of the sensitive attribute  $\psi$  in the  $j^{th}$  event of the sequence  $S_i$  is denoted by  $e_{ij}(\psi)$ . A subset of attributes  $\{A_1, A_2, \dots, A_n\}$  is assumed to be publicly available, so they act as quasi-identifiers,  $QIs \subseteq \{A_1, A_2, \dots, A_n\}$ . The values of the sensitive attribute are naturally private. We assume an adversary who knows that the record of a target individual exists in a released multidimensional sequence dataset. She also has some background knowledge about the sequential events of a target individual, i.e. the values of some  $QIs$  as well as the order of these values in some of the events of an individual's sequence. Armed with this knowledge, the adversary seeks to find some *matching* records to her background knowledge in the released data. If the number of such records is not "sufficiently" large or the percentage of sequences among these records containing a common sensitive value  $\sigma$  is high, the adversary may infer some sensitive information about the individual. Since adversaries'

knowledge is assumed to be in the form of any combination of  $QIs$ ' values, the worst-case scenario would be an adversary knowing the values of *all*  $QIs$ ' in *all* events of a target individual. Therefore, to protect privacy of individuals the privacy model should ensure that every sequence in the released data is linked to a sufficiently large number of other sequences and the percentage of sequences with the same sensitive value in every group of indistinguishable sequences is not too high. However, the latter case may not need to be satisfied for every value of the sensitive attribute. More precisely, if some values of the sensitive attribute have less degree of sensitivity and do not need to be kept private, then we do not need to be worried about these values being too frequent in a group. For example, in the context of publishing medical data, it might be allowed to disclose the value "flu" for the sensitive attribute disease. To effectively handle these cases, we define a set  $\Omega \subseteq \Psi$ , called *highly-sensitive* set, which contains those values of the sensitive attribute  $\Psi$  which have a high degree of sensitivity. In the presence of this set, our privacy model must ensure that the frequency of sequences which have at least one of the values in  $\Omega$  in some of their events is not too high in any group of indistinguishable sequences. This brings us to the following definition.

**DEFINITION 1** ( $(k, c)$ -privacy). Given anonymity threshold  $k \geq 2$ , and confidence threshold  $c \in (0, 1]$ , a multidimensional sequence dataset  $D$  satisfies  $(k, c)$ -privacy if *i*) each sequence in  $D$  is indistinguishable from at least  $k-1$  other sequences with respect to any combination of  $QIs$  and *ii*) the probability of inferring any high sensitive value in any group of indistinguishable sequences is at most  $c$ .

## 2.2 Information Loss

We employ generalization and suppression on the values of  $QIs$  to modify data and form clusters. This anonymization process incurs information loss because some original values of  $QIs$  in every sequence are either replaced with less specific values or are totally removed. In order to preserve data utility for data mining tasks, we should ensure that anonymization cost is minimized. We consider the scenario where the data analysis task is unknown at the time of data publication. So, our goal is to anonymize a multidimensional sequence data to satisfy  $(k, c)$ -privacy while preserving data utility as much as possible. Let  $D^*$  be an anonymization of the multidimensional sequence data  $D$ .  $D^*$  corresponds to a set of clusters  $C = \{C_1, C_2, \dots, C_p\}$  which is a clustering of sequences in  $D$ . All sequences in a given cluster  $C_j$  are anonymized together. We define the amount of information loss incurred by anonymizing  $D$  to  $D^*$  as

$$IL(D, D^*) = \frac{1}{|D|} \sum_{j=1}^p IL(C_j) \quad (1)$$

where  $IL(C_j)$  is the information loss of the cluster  $C_j$ , which is defined as the sum of information loss of anonymizing every sequence  $S$  in  $C_j$ :

$$IL(C) = \sum_{i=1}^{|C|} IL(S_i, S_i^*) \quad (2)$$

where  $|C|$  is the number of sequences in the cluster  $C$ , and  $IL(S, S^*)$  is the information loss of anonymizing the sequence  $S$  to the sequence  $S^*$ .

Each sequence is anonymized by generalizing or suppressing some of the  $QIs$ ' values in some of its events. So, we define information loss of a sequence based on the information loss of its events. Let  $H$  be generalization hierarchy of the attribute  $A$ . We use the *Loss Metric (LM)* measure [3] to capture the amount of information loss incurred by generalizing the value  $a$  of the attribute  $A$  to one of its ancestors  $\hat{a}$ , with respect to  $H$ :

$$IL(a, \hat{a}) = \frac{|\mathcal{L}(\hat{a})| - |\mathcal{L}(a)|}{|\Delta_A|} \quad (3)$$

where  $|\mathcal{L}(x)|$  is the number of leaves in the subtree rooted at  $x$ .

The information loss of each event  $e$  is then defined as

$$IL(e, e^*) = \sum_{n=1}^{|QI|} IL(e(n), e^*(n)) \quad (4)$$

where  $e^*$  is the ancestor of the event  $e$ ,  $e(n)$  is the value of  $n^{th}$   $QI$  of the event  $e$  and  $e^*(n)$  is its corresponding value in the event  $e^*$ .

Hence, the information loss incurred by anonymizing each sequence is as follows:

$$IL(S, S^*) = \sum_{m=1}^{|S|} IL(e_m, e_m^*) \quad (5)$$

## 3. ANONYMIZATION ALGORITHM

We propose a bottom-up anonymization algorithm based on hierarchical agglomerative clustering. The general idea is to anonymize data by starting with the trivial clustering that consists of singleton clusters and then keep merging the two closest clusters, until all clusters satisfy privacy constraints based on  $(k, c)$ -privacy model. A key factor in any clustering algorithm is the distance measure. In order to minimize the overall data distortion due to anonymization, we define the distance between two given clusters as the change in information loss when we merge the clusters:

$$dist(X, Y) = IL(X \cup Y) - IL(X) - IL(Y) \quad (6)$$

where  $IL(X \cup Y)$  is the information loss of the merged cluster, and  $IL(X)$  and  $IL(Y)$  are information loss of clusters  $X$  and  $Y$  before merge, respectively.

We assume that every cluster has a representative sequence which is the result of anonymizing all contained sequences. The distance between two clusters is calculated based on the information loss of anonymizing their representatives, and the clusters with the smallest distance are chosen to be merged. In general, two representative sequences have different number of events. So, the anonymization of these sequences can be seen as the problem of finding a matching between the events of these sequences, using generalization and suppression, such that the anonymization cost is minimized. The following definition expresses the information loss of a merged cluster based on the information loss of anonymizing representatives of two clusters which are being merged to their best matching.

**DEFINITION 2.** Let  $\tilde{X}$  and  $\tilde{Y}$  be representative sequences of clusters  $X$  and  $Y$  and  $M_{XY}$  be their best matching. Then the information loss of the merged cluster  $X \cup Y$  is define as

$$IL(X \cup Y) = |X| \cdot IL(\tilde{X}, M_{XY}) + |Y| \cdot IL(\tilde{Y}, M_{XY}) \quad (7)$$

where  $IL(\tilde{X}, M_{XY})$  and  $IL(\tilde{Y}, M_{XY})$  are information loss of anonymizing representative sequences  $\tilde{X}$  and  $\tilde{Y}$  to their best matching sequence  $M_{XY}$ .

Finding the best matching between two sequences is a *sequence alignment* problem. The basic principle underlying sequence alignment methods is to measure the effort it takes, in terms of specific operations, to make sequences equal. One of the most common approaches for sequence alignment is *dynamic programming*. Dynamic programming is an advanced algorithmic technique that solves optimization problems from the bottom up by finding optimal solutions to subproblems. Inspired by [7], we

employ dynamic programming to align representatives of clusters with the goal of minimizing anonymization cost. The operations which we use to align (anonymize) two sequences are generalization and suppression. If two values of the attribute  $q \in QI$  are identical, their generalization is equal to the values themselves; otherwise both values are replaced with their *lowest common ancestor (LCA)* which is the lowest node in the generalization hierarchy  $H_A$  that is an ancestor of both  $v$  and  $w$ . Given sequences  $\tilde{X} = \{x_1, x_2, \dots, x_t\}$  and  $\tilde{Y} = \{y_1, y_2, \dots, y_t\}$  as the representatives of two clusters, the optimal alignment of these two sequences is the alignment which incurs minimum information loss considering all  $QIs$ . We have three cases for aligning  $\tilde{X}$  and  $\tilde{Y}$ : 1) aligning  $\{x_1, x_2, \dots, x_{t-1}\}$  and  $\{y_1, y_2, \dots, y_{t-1}\}$ , and generalizing  $x_t$  and  $y_t$ , which means replacing every  $QI$  value in  $x_t$  and its corresponding  $QI$  value in  $y_t$  with their *LCA*, 2) aligning  $\{x_1, x_2, \dots, x_{t-1}\}$  and  $\{y_1, y_2, \dots, y_t\}$ , and suppressing  $x_t$ , 3) aligning  $\{x_1, x_2, \dots, x_t\}$  and  $\{y_1, y_2, \dots, y_{t-1}\}$ , and suppressing  $y_t$ .

For every  $q \in QI$  we create a score matrix to store the cost of all sub-problems for aligning two one-dimensional sequences resulted from projecting sequences  $\tilde{X}$  and  $\tilde{Y}$  on  $q$ . Each of these solutions have an anonymization cost and our objective is to find the best alignment with minimum information loss. The cost of each solution is calculated as the sum of its cost for every  $q \in QI$ . Besides the score matrices, we assume a *move* matrix  $M$  where each cell  $M[i, j]$  contains the operation which is chosen to align the sequence prefix  $x_1, x_2, \dots, x_i$  and the sequence prefix  $y_1, y_2, \dots, y_j$ . To build the sequence  $M_{X,Y}$  which is the result of best alignment of sequences  $\tilde{X}$  and  $\tilde{Y}$ , we do a “traceback” on matrix  $M$  from cell  $M[t+1, t+1]$  to cell  $M[0, 0]$ .

Our clustering algorithm *clustering based multidimensional sequence data anonymizer (CBMSA)* is based on agglomerative hierarchical clustering. We start with the trivial case of singleton clusters and iteratively merge two closest clusters which are determined by applying our multidimensional sequence alignment algorithm. Once, a cluster satisfies  $(k, c)$ -privacy, it will not be merged anymore. In order to reduce information loss, our preference is to merge two closest clusters which do not violate the confidence constraint of  $(k, c)$ -privacy model when being merged. When we merge two clusters  $X$  and  $Y$  with representative sequences  $\tilde{X}$  and  $\tilde{Y}$ , all sequences in clusters  $X$  and  $Y$  are anonymized with respect to  $M_{X,Y}$ . This means that those events which are suppressed in  $\tilde{X}$  and  $\tilde{Y}$  based on the best alignment result are suppressed in all sequences in clusters  $X$  and  $Y$ , respectively. The remaining events of every sequence are then replaced with their corresponding events in  $M_{X,Y}$ . However, since we only apply generalization on  $QI$  values, the values of sensitive attribute in these events remain unchanged. Since our goal is to build clusters which satisfy  $(k, c)$ -privacy, for every cluster we should check if it contains at least  $k$  sequences and if the frequency of sequences which have at least one event with a high sensitive value is not greater than  $c$ . When we merge two clusters  $X$  and  $Y$ , the size of the new cluster is simply the sum of the number of sequences in  $X$  and  $Y$ . For the diversity check, we should count the number of sequences which have at least one event with a high sensitive value. In order to efficiently count these sequences, for every cluster we use a data structure, denoted by *HighSensList*, to keep track of these sequences. When we merge two clusters  $X$  and  $Y$ , the number of sequences with high sensitive value in  $X$  or  $Y$  may decrease. This is due to the fact that some events may be suppressed in sequences of cluster  $X$  or  $Y$ . If the events which are suppressed in a sequence are the only ones which contain high sensitive value, then this sequence will not

contain any high sensitive value after applying suppression. So it should be removed from *HighSensList* of the cluster where it is consist of. So, after applying anonymization on sequences of clusters  $X$  and  $Y$ , we first update *HighSensList* of these clusters and then merge two *HighSensLists* to build the *HighSensList* of the new merged cluster. We keep merging clusters till no more than one cluster left. If the remained cluster does not satisfy privacy constraints, we remove all sequences contained in this cluster from data.

## 4. EXPERIMENTS

In this section, our goal is to evaluate the performance of our proposed anonymization algorithm in terms of information loss calculated based on Equation 1 as well as scalability by varying the anonymity threshold  $k$  and the confidence threshold  $c$ . We developed a data generator to generate synthetic multidimensional sequence data inspired from the Heritage Health Prize (*HHP*) claims data set<sup>1</sup>. In [2], anonymization of the *HHP* claims dataset is studied in order to prevent identity disclosure attacks. Authors identified 6  $QIs$  for the claims dataset among which we selected two attributes *days since first claim in each year (DSFC)* and *length of stay (LOS)*. We did not include attributes *diagnosis* and *CPTCode*<sup>2</sup> since these attributes are sensitive attributes in the framework of our study. Also, we disregarded attributes *place of service* and *specialty* due to not accessing to their possible original values in the *HHP* data. Instead, for each claim we included two other  $QI$  attributes, i.e. *ZIP code* of patients due to the fact that this information is often updated at every visit and the year in which a claim took place (*AdmYr*). We generated multiple synthetic datasets by varying the number of sequences, average number of events per sequence (3, 5, and 10), and number of  $QIs$  per event (2, 3, and 4). For every set of data characteristics, we generated 10 datasets, evaluated their performance in terms of information loss, and took the average of information loss of 10 datasets in each set. We implemented our algorithms in Java and conducted experiments on a 1.80 GHz Intel core i5 PC with 8 GB RAM. To illustrate the benefits of our proposed multidimensional sequence alignment method, we also developed a baseline algorithm which does not use dynamic programming. If two sequences  $X$  and  $Y$  are of the same size, the baseline algorithm simply applies generalization to every event of two sequences. Otherwise, it first randomly suppresses  $n = \text{abs}(|X| - |Y|)$  events in the longer sequence and then generalizes every remaining events in two sequences. In the first set of experiments, we evaluate the information loss  $IL$  by varying the value of the anonymity threshold  $k$  while keeping the confidence threshold  $c$  fixed. Figure 2 shows the  $IL$  for two datasets of size 1000 and 10000 with the average number of events 5 and three  $QIs$  with anonymity threshold  $5 \leq k \leq 50$  and a fixed confidence threshold  $c = 0.7$ . As  $k$  increases,  $IL$  increases for both algorithms. This illustrates the trade-off between privacy and data utility. In other words, as  $k$  increases, higher level of privacy protection is required to keep the probability of re-identifying a target individual or inferring sensitive information about a target individual fairly low. Therefore, more data distortion occurs. Also, comparing the information loss of our anonymization algorithm based on dynamic programming with the baseline algorithm depicts the benefits of our method. In the second set of experiments we change  $c$  from 0.5 to 0.9 for the fix value  $k = 5$ . This setting allows us to measure the performance of our anonymization

<sup>1</sup> <http://www.heritagehealthprize.com/c/hhp/data>

<sup>2</sup> Current Procedural Terminology Code



algorithm against attribute disclosure for a fixed  $k$ . The resulting information loss of two algorithms for two datasets is shown in Figure 3. In general,  $IL$  decreases as  $c$  increases due to a less restrictive privacy requirement. Similar trends were observed between CBMSA and Baseline for the other datasets. The results are omitted for brevity. In Figure 4, we show the time performance of our algorithm for two datasets with  $5 \leq k \leq 50$  and a fixed confidence threshold  $c=0.7$ . For a small dataset with total number of events about 4500, for every value of  $k$ , the total runtime of our algorithm is less than 30 sec. For a large dataset, with total number of events about 47000, the execution time of our algorithm for different values of  $k$  is between 2200 sec and 2700 sec. The run time of baseline algorithm for both datasets is very fast. This indicates that our algorithm spends a large amount of its running time on multidimensional sequence alignment based on dynamic programming. Even though the run time of our algorithm for large datasets is fairly high, we believe it is still acceptable in practice due to the fact that most anonymization tasks are off-line procedures.

## 5. CONCLUSIONS AND FUTURE WORKS

In this paper, we proposed an anonymization algorithm for multidimensional sequence data using sequence alignment techniques and agglomerative hierarchical clustering. To the best of our knowledge, this is the first work for multidimensional sequence data anonymization which prevents both identity disclosure and attribute disclosure without making any assumption about the knowledge of the adversary. Our experimental results on synthetic data show the effectiveness of our proposed algorithm for anonymizing multidimensional sequence data. Our future work includes the following. In this work we assumed that the goal of data publication is unknown and we anonymized data by minimizing data distortion for general data analysis purposes. In our future work, we will consider the case of publishing data for a specific data mining task such as classification. This requires employing an appropriate anonymization cost measure to capture the utility of our algorithm for data mining tasks. Moreover, in this paper, we studied the simplest case of a single sensitive attribute in every event of a sequence. An extension of our work would be the case of multiple sensitive attributes. Also, we will run experiments on real datasets to further investigate the effectiveness of our proposed algorithm.

## Acknowledgment

The authors would like to thank Dr. Khaled El Emam (EHIL lab) for his inspiration with this research. The early stages of this work were partially supported by grants of CIHR and NSERC.

## 6. REFERENCES

- [1] El Emam, K., Arbuckle, L., Koru, G., Gaudette, L., Neri, E., Rose, S., Howard, J., and Gluck, J., 2012. De-Identification Methods for Open Health Data: The Case of the Heritage Health Prize Claims Data Set. *In Journal of Medical Internet Research*, 14:1, DOI:10.2196/jmir.2001, 2012.
- [2] Fung, B.C.M, Wang, K, Chen, R. and Yu, P.S. 2010. Privacy-preserving data publishing: A survey of recent developments. *ACM Comput. Surv.* 42, 4, Article 14 (June 2010), 53 pages
- [3] Iyengar, V.S. 2002. Transforming data to satisfy privacy constraints. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 279-288.

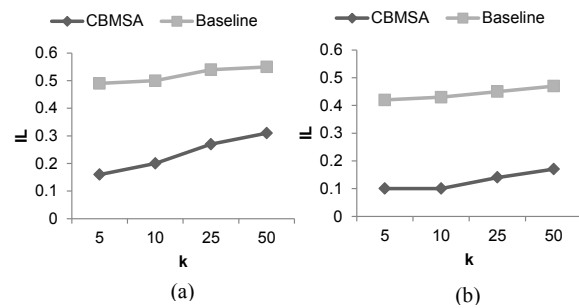


Figure 2. Information loss for (a) Data\_1000\_5\_3 (b) Data\_10000\_5\_3 with  $c = 0.7$

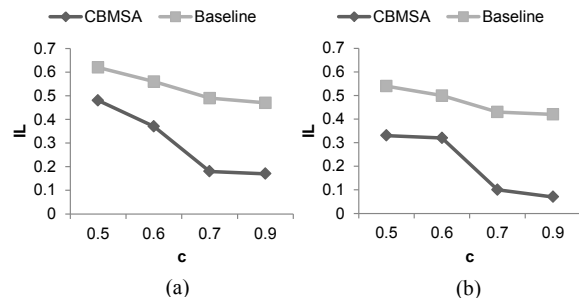


Figure 3. Information loss for (a) Data\_1000\_5\_3 (b) Data\_10000\_5\_3 with  $k = 5$

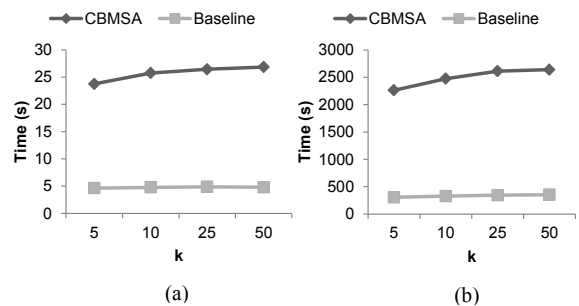


Figure 4. Execution time for (a) Data\_1000\_5\_3 (b) Data\_10000\_5\_3 with  $c = 0.7$

- [4] Kaufman L and Rousseeuw, P. J., 1990. Finding Groups in Data: An Introduction to Cluster Analysis. *John Wiley* 1990
- [5] Needleman, S. B. and Wunsch, C. D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Molecular Biol.*, vol. 48, no. 3, pp. 443-453, 1970.
- [6] Sehatkar, M. and Matwin S., 2013. HALT: Hybrid Anonymization of Longitudinal Transactions, Eleventh Annual International Conference on Privacy, Security and Trust (PST), Tarragona, Spain.
- [7] Tamersoy, A., Loukides, G., Nergiz, M. E., Saygin, Y. and Malin, B. 2012. Anonymization of Longitudinal Electronic Medical Records. *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, pp. 413-423, 2012

# Efficient Multi-User Indexing for Secure Keyword Search

Eirini C. Micheli, Giorgos Margaritis, Stergios V. Anastasiadis  
Department of Computer Science and Engineering  
University of Ioannina, Greece  
{emicheli,gmargari,stergios}@cs.uoi.gr

## ABSTRACT

Secure keyword search in shared infrastructures prevents stored documents from leaking confidential information to unauthorized users. We assume that a shared index provides confidentiality if it can only be used by users authorized to search all the documents contained in the index. We introduce the Lethe indexing workflow to improve query and update efficiency in secure keyword search. Lethe clusters together documents with similar sets of authorized users, and only creates shared indices for configurable volumes of documents with common users. Based on the published statistics of an existing dataset, we show that Lethe generates an indexing organization that simultaneously achieves both low search and update cost.

## Categories and Subject Descriptors

H.3 [Information Storage and Retrieval]: Information Search and Retrieval; K.6 [Management of Computing and Information Systems]: Security and Protection

## General Terms

Design, Experimentation, Measurement, Security

## Keywords

inverted index, clustering, full-text search, shared data storage, confidentiality

## 1. INTRODUCTION

Keyword (or full-text) search is an indispensable service for the automated retrieval of text documents, whether proprietary within an organization, or public across the web. Over the years, an enormous amount of accumulated text has gradually expanded keyword search to several contemporary storage environments, such as personal content archives, online social networks, and cloud facilities. At the same

time, the efficiency benefits of storage consolidation increasingly motivate the maintenance of sensitive data over public infrastructures. Indeed, the access control enforced at the storage level is often presumed sufficient for the necessary confidentiality isolation of co-located users and organizations.

An inverted index is the dominant indexing structure in keyword search. The stored documents are preprocessed into a posting list per keyword (or term), which provides the occurrences (or postings) of the term across all the documents. A single index shared among multiple users offers search and storage efficiency. However, it can also leak confidential information about documents with access permissions limited to a subset of the users [5, 13, 10, 3]. The problem persists even if a query is initially evaluated over the shared index, and later the inaccessible documents are filtered out from the final result list before it is returned to the user [5].

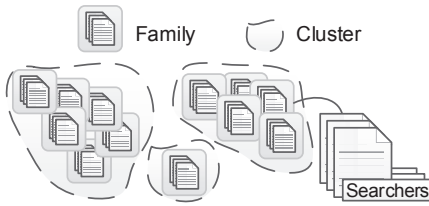
A known secure solution applies a shared index by limiting search to term postings of documents searchable by the user [5]. During query processing it skips dependencies on inaccessible documents through posting filtering at extra list processing overhead. In online social networks, recent research applies advanced list-processing operators and cost models to improve secure search efficiency [3]. First, it organizes the friends of each user into appropriate groups based on characteristics of the search workload. Then, during query handling, it intersects the list of documents that contain a term against the list of documents authored by the querying user and the union of her friend groups.

A different secure solution partitions the document collection by search permissions, and maintains a separate index for each partition [13]. The collection ends up indexed by a limited number of indices, and query handling runs over all the indices that contain documents searchable by the querying user. However, minor variations in search permissions of different documents increases the number of indices. Although smaller indices can be completely eliminated by replicating their contents to private per-user indices, this approach increases document duplication across the indices and the respective update cost.

In this study, we aim to achieve low search latency and index update cost by limiting both the number of indices per user and the document duplication across the indices. We group by search permissions the documents into families, and cluster together the families with similar permissions. We maintain one index for the documents searchable by a maximal common subset of users in a cluster. Cluster docu-

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

*7th International Workshop on Privacy and Anonymity in the Information Society (PAIS'14)* March 28, 2014, Athens, Greece



**Figure 1: Document families grouped by searcher similarity  $L_s$  into clusters.**

ments whose users lie outside the above subset are inserted into either per-user private indices or additional multi-user indices.

Our indexing organization for secure keyword search is innovative because we (i) skip query-time list filtering via prebuilt securely-accessible indices, and (ii) effectively reduce the number of searched or maintained indices through configurable partial merging of indices for documents with common authorized users. In Sections 2 and 3 we present the Lethe indexing workflow and our prototype implementation. In Sections 4 and 5 we show some experimental results and examine previous related work, while in Section 6 we summarize our conclusions and plans for future work.

## 2. INDEXING ORGANIZATION

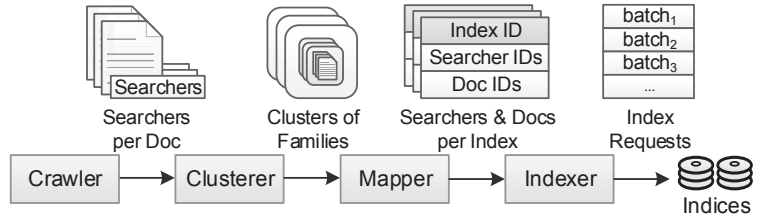
We next provide the basic assumptions and goals of our work, and describe the stages of the Lethe indexing workflow that we propose.

### 2.1 Assumptions and Goals

We target collections of text documents in shared storage environments accessible by multiple users. The system applies access control to protect the confidentiality and integrity of the stored documents from actions of unauthorized users. We designate as *owner* the user who creates a document, and *searchers* of the document the users who are authorized to search by keywords for the document and read its contents. The system preprocesses the documents content into the necessary indexing structure to enable interactive search through keyword criteria set by the searchers. In our indexing organization we set the following goals:

- **Security** Ensure that the indexing structure provides confidentiality of the searched documents with respect to the document contents and their statistical characteristics (e.g., number of documents, term properties).
- **Search Efficiency** Minimize the search latency per query as measured through an appropriate metric (e.g., median or high percentile).
- **Indexing Cost** Minimize the document insertion I/O activity and indexing storage space required for the entire collection.

We require that users are authenticated by the system and authorized to only search documents with the necessary access permissions. Accordingly, we build a separate index for each document subset with common access permissions. We presently examine secure search in multi-user environments, but leave outside the study scope the closely related but complementary problem of search over encrypted storage. In fact, search with encrypted keywords over encrypted



**Figure 2: The four stages of the Lethe workflow.**

documents conceals the search activity and stored documents from a storage provider, but it does not necessarily hide the characteristics of stored content from unauthorized searchers [13].

### 2.2 The Lethe workflow

We introduce the Lethe workflow consisting of four basic stages for crawling, clustering and mapping the documents to the generated indices.

**Crawler** In order to realize our goals, we build an appropriate indexing organization based on the document search permissions. Let a text dataset  $\mathcal{T} = (D_{\mathcal{T}}, S_{\mathcal{T}})$ , where  $D_{\mathcal{T}}$  is the set of all documents, and  $S_{\mathcal{T}}$  the set of all users with search permissions over one or more documents of  $D_{\mathcal{T}}$ . First we crawl the names (e.g., paths) and permissions (e.g., allowed searchers) of documents in  $\mathcal{T}$ , and assign unique identifiers to the members of  $D_{\mathcal{T}}$  and  $S_{\mathcal{T}}$ . Then we group into a separate *family*  $f = (D_f, S_f)$ , each set of documents  $D_f \subseteq D_{\mathcal{T}}$  with identical set of searchers  $S_f \subseteq S_{\mathcal{T}}$ .

**Clusterer** We aim to maintain a single index for the searchers who are common among similar families. Accordingly, we need to identify those families with substantial overlap in their searcher sets. We address this issue as a universal clustering problem over the searcher sets of the families in the entire dataset (Fig. 1). We parameterize the clustering method as necessary to assign every family to exactly one cluster, without omitting any families as noise.

Let the *searcher similarity*  $L_s \in [0, 1]$  be a configurable parameter to adjust the number of common searchers across the families of each created cluster. We generate a set  $C_{\mathcal{T}}$  of clusters, where each cluster  $c \in C_{\mathcal{T}}$  contains a set  $F_c$  of families, and each family  $f \in F_c$  contains the document set  $D_f \subseteq D_{\mathcal{T}}$ . The document set  $D_c$  of cluster  $c$  is derived from the union of the documents contained across all the families of  $c$ , i.e.,  $D_c = \bigcup_{f \in F_c} D_f$ . Thus, the number of documents in cluster  $c$  is at least as high as the number of families in  $c$ , i.e.,  $|D_c| \geq |F_c|$ .

**Mapper** We strive to map each family  $f$  to the minimum number of indices required to securely handle keyword queries over the documents in  $D_f$ , but also minimize the total number of indices in the system. First, we dedicate to every searcher  $u \in S_{\mathcal{T}}$  the pair  $P_u = (D_u^e, \{u\})$ , where  $D_u^e \subseteq D_{\mathcal{T}}$  is the set of documents exclusively searchable by  $u$ . Then, we assign to  $P_u$  a *private* index  $I_u$  containing the documents of  $D_u^e$ .

Let the *cluster intersection*  $P_c$  of cluster  $c \in C_{\mathcal{T}}$  be a pair  $(D_c^i, S_c^i)$ , with  $D_c^i = D_c$ , and  $S_c^i = \bigcap_{f \in F_c} S_f$  the intersection of searchers in the families of  $F_c$ . By family definition, the documents in  $D_c^i$  are searchable by all the searchers in  $S_c^i$ . If  $|S_c^i| \neq \emptyset$ , we dedicate a separate index  $I_c$  to the intersection  $P_c$ . For every family  $f \in F_c$ , we also define a *family*

difference  $P_f$  as the pair  $(D_f^d, S_f^d)$ , where  $D_f^d = D_f$  and  $S_f^d = S_f - S_c^i$ , i.e.,  $S_f^d$  corresponds to the searchers of family  $f$  not contained in  $S_c^i$  of  $P_c$ . If  $S_f^d \neq \emptyset$ , we have to allow the users  $u \in S_f^d$  to securely search for documents  $d \in D_f^d$ .

An extreme approach to address the above  $P_f$  search problem is to insert every document  $d \in D_f^d$  to every private index  $I_u, u \in S_f^d$ . However, a difference  $P_f$  may contain a relatively large number  $|D_f^d|$  of documents searchable by a considerable number  $|S_f^d|$  of users. Hence, the above approach would end up to a large number of documents duplicated across the private indices of many users. At the other extreme, we could dedicate a separate index  $I_f$  to every difference  $P_f$  with  $|S_f^d| \neq \emptyset$ . However, this approach runs the risk of generating in the system a large number of indices, each serving a small number of documents and searchers.

We introduce the *duplication product*  $R_f^d = |D_f^d| \cdot |S_f^d|$  to approximate<sup>1</sup> the potential document duplication resulting from indexing a family difference  $P_f$ . Subsequently, the decision of whether we should create a dedicated index  $I_f$  depends on how  $R_f^d$  compares to the configurable *duplication threshold*  $T_d$ . We assume that  $R_f^d < T_d$  implies an affordable cost of inserting the documents  $d, \forall d \in D_f^d$ , to private indices  $I_u, \forall u \in S_f^d$ . Instead,  $R_f^d \geq T_d$  suggests that devoting a separate index  $I_f$  to the difference  $P_f$  is preferable.

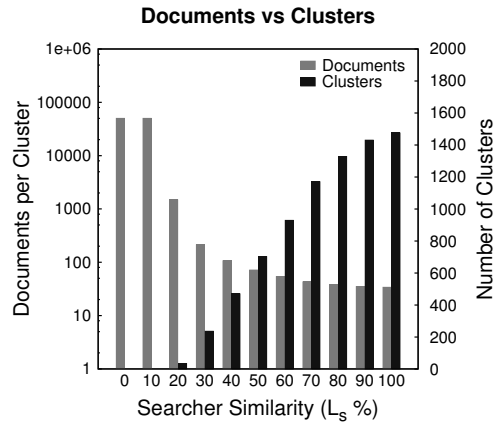
An optimization that we do not examine further due to its complexity is to pursue additional duplication reduction by intersecting the searchers of the differences  $P_f, \forall f \in F'_c$ , for appropriate  $F'_c \subset F_c$  corresponding to cluster  $c$ .

**Indexer** We insert each document  $d \in D_{\mathcal{T}}$  to the appropriate  $I_c, I_f$ , and  $I_u$  indices specified by the above mapping phase. In order to keep low the necessary I/O activity, we separately generate each index through a specification of the contained documents. We experimentally validated that the alternative approach of specifying to the system the indices of each document leads to higher I/O activity due to lower storage locality during the index updates. As new documents are added to the collection, we look for existing indices to securely serve all the searchers of each document. Periodically, we repeat the previous clustering and mapping phases to optimize the search over the accumulated document collection. Deletions or modifications of inserted documents are handled with the necessary changes of the index contents and potential reorganization of their mapping to documents. We summarize the four stages of the Lethe workflow along with their outputs in Fig. 2.

### 3. PROTOTYPE IMPLEMENTATION

Based on the above design, our prototype implementation consists of four components: (i) *crawler*, (ii) *clusterer*, (iii) *mapper*, and (iv) *indexer*. The crawler specifies a unique identifier for each document and gathers information about the permitted document searchers. The clusterer organizes the documents into families according to their searchers, and then clusters the families based on the searcher similarity  $L_s$ . We use the searchers of each document as key to create the families over a hash table. Thus, all documents with identical searchers end up at the same entry of the table.

<sup>1</sup>For increased accuracy of  $R_f^d$  over diverse document sizes, we could replace  $|D_f^d|$  with the total number of postings contained in all documents  $d \in D_f^d$ .



**Figure 3:** For the synthetic dataset based on DocuShare[14], we examine the number of created clusters and the number of documents per cluster across different  $L_s$  values.

Subsequently, we group the families with similar searchers into the same cluster represented as a vector of family identifiers. The searchers of a family  $f$  are concisely represented through a *searcher bitmap*  $M_f$  of length equal to the number of users  $|S_{\mathcal{T}}|$  in the stored dataset. In bitmap  $M_f$  we set equal to 1 the values at bit positions specified by identifiers of permitted family searchers  $u \in S_{\mathcal{T}}$ .

Since we do not know in advance the number of clusters, we use a clustering algorithm that produces this number as output (e.g., DBSCAN) rather than requiring it as input (e.g., K-means) [16]. Within each cluster, the mapper identifies the cluster intersections and family differences. Each intersection or difference is specified through the contained documents and authorized searchers. We assign a dedicated index to each cluster intersection, and we use a dedicated index or the private indices of the respective searchers for each family difference according to the duplication threshold  $T_d$ . The indexer receives the index specifications from the mapper, and splits each index into document batches. Then it communicates with the search engine to insert the documents of each batch to the respective index, after initializing it if necessary. Finally, the search engine serves queries by using the indices permitted to each authorized searcher.

## 4. EXPERIMENTAL EVALUATION

We use the published statistics of a real dataset to generate a synthetic workload, and apply a prototype implementation of the Lethe workflow that we developed. Then we measure the number of indices per user and document for different parameters, and analyze the security and efficiency characteristics of our approach.

### 4.1 Document Dataset

We generate a synthetic document collection with searcher lists based on published measurements of an existing dataset (DocuShare [14]). We set the number of users to 200, user groups to 131, documents to 50000, and max group size to 50. We specify the sizes of individual groups from the DocuShare statistics, and uniformly pick users as group members. Based on the DocuShare statistics, we specify the number of users and groups allowed to search each docu-

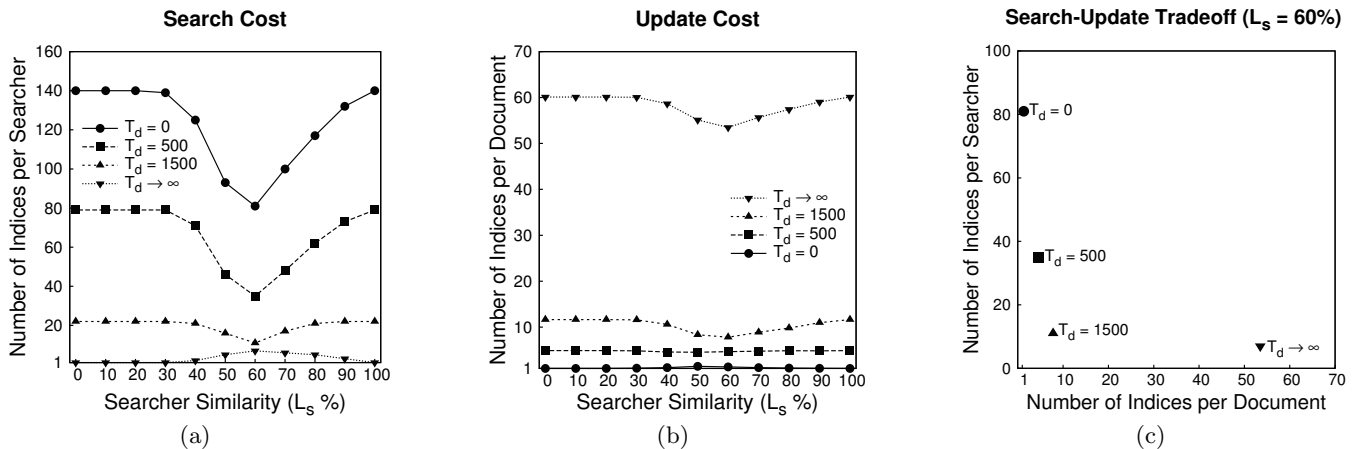


Figure 4: For the synthetic dataset based on DocuShare[14], we illustrate the number of indices (a) per searcher and (b) per document across different  $L_s$  and  $T_d$  values. In addition, we show (c) the search-update tradeoff for different  $T_d$  values at fixed  $L_s=60\%$ .

ment, and then uniformly assign to each document specific users and groups. We implemented the crawler, clusterer and mapper in C/C++ with STL, and the indexer in Perl (v5.10.1). For clustering we applied the DBSCAN algorithm with  $\text{MinObjs}=1$  and  $\text{Eps}=L_s$  [16]. We run the computations over Linux v2.6.32 on quad-core x86 2.33GHz processor, 4GB RAM, and 7.2KRPM SATA disks.

## 4.2 Measurement Results

We applied the Lethe workflow to organize the examined dataset into clusters of document families. For different  $L_s$  values, in Fig. 3 we show the average number of documents per cluster and the total number of clusters. The duplication threshold  $T_d$  is not included because it only applies to the subsequent mapping stage. The ideal similarity should result into family clusters with common searchers per cluster to be efficiently served by a single index. For instance, setting  $L_s=60\%$  generates 929 clusters with 53.82 documents per cluster. At the extreme case of  $L_s=0\%$ , there is 1 cluster containing all 1475 families and 50000 documents. At the other extreme of  $L_s=100\%$ , there are 1475 clusters, each containing 1 family with 33.90 documents on average.

We regard the number of indices per searcher as a proxy of the search cost, because it specifies the number of document lists that have to be merged into the final search result. Accordingly, in Fig. 4a we examine the sensitivity of the search cost to the values of the  $L_s$  and  $T_d$  parameters. We experimented with  $T_d$  values in the range  $[0, +\infty)$ . The indices per searcher vary between 35 and 79 at  $T_d=500$ , and between 11 and 22 at  $T_d=1500$ . Setting  $L_s=0\%$  or  $100\%$  usually maximizes the number of indices per searcher. This follows from the fact that index sharing is limited in a single cluster of diverse families, or numerous clusters of one family each. On the contrary, setting  $L_s=60\%$  leads to non-empty cluster intersections, and roughly minimizes the number of indices per searcher. One exception to the above pattern occurs with  $T_d \rightarrow \infty$ , which prohibits index sharing within family differences, and minimizes the indices per searcher at  $L_s=0\%$  or  $100\%$  instead of  $L_s=60\%$ .

The update cost of the indexing organization can be proxied through the average number of indices that contain each

document, and have to be updated during document insertion. In Fig. 4b we examine the sensitivity of the update cost to  $L_s$  and  $T_d$ . At  $L_s=60\%$ , we notice that setting  $T_d=1500$  or  $T_d \rightarrow \infty$  minimizes the number of indices per document to 7.80 and 53.48, respectively. Instead, the curves remain almost flat across different  $L_s$  values when  $T_d=0$  or 500. If we combine this observation with the outcome of the previous paragraph, we conclude that  $L_s=60\%$  leads to both low update and search cost.

A striking difference between Figures 4a and 4b is the opposite effect of  $T_d$  to the search and update cost. This tradeoff is further illustrated in Fig. 4c for different  $T_d$  values and fixed  $L_s=60\%$ . We found  $T_d=1500$  to provide a reasonable choice, because it simultaneously achieves a low number of 11 indices per searcher and 7.8 per document. Overall, at  $L_s=60\%$  and  $T_d=1500$ , the mapper specifies a total of 298 indices: 84 and 182 shared indices for intersections and differences, respectively, and 32 private indices. In early measurements (not shown) that we did over a search engine, the above results directly translated to low search and update latency unlike alternative settings.

## 4.3 Analysis of Results

Our preliminary experiments provide strong evidence for an improved method to achieve efficient and secure keyword indexing. The method is secure because a query can only use indices of documents that the searcher is permitted to access [5]. The method is also efficient for several reasons.

First, we guarantee that the result returned by an index does not require any filtering to remove documents inaccessible to the searcher. We only require to merge the results from multiple indices for ranking purposes, as is typically already done by parallel or distributed search engines. Thus, we avoid the extra query-time overhead for list processing required by previous secure methods [3].

Second, the clustering of document families allows the service of common searchers in the cluster intersection with a single index. Thus, we reduce the average number of indices per searcher, which translates into smaller number of result lists to be generated and merged during query handling. To the best of our knowledge, this is the first time that cluster-

ing is applied for the efficiency of secure keyword search.

Third, the control of indexing duplication through the threshold  $T_d$  prevents the insertion of the same document to an excessive number of multiple private indices, which was previously required [13]. Instead, we create extra shared indices whenever the number of documents and their common searchers justify their cost.

## 5. RELATED WORK

We compare our work with related research results previously developed for secure text indexing, remote storage of encrypted documents, and online social networks.

**Security-aware Indexing** Büttcher and Clarke examine the problem of filesystem search with relevance ranking based on the vector space model [5]. A secure search engine must only deliver query results dependent on files searchable by the querying user. Thus, a system-wide index to find and rank all matching files is insecure, because it can leak the total number of files matching a term, or term statistics normally unavailable to a user. As a solution, the authors propose to restrict query processing to the parts of posting lists that the querying user is permitted to access. The resulting performance slowdown can be reduced through appropriate reordering of query operators.

Singh et al. logically organize the filesystem into sets of files, called access-control barrels, with identical access privileges of users and groups [13]. The system constructs a separate index per barrel, and restricts query handling to permitted barrels. The authors define the access credentials of users, groups and barrels, and use them as nodes of the access credentials graph. The graph includes edges that minimally connect users to their groups and searchable barrels. The authors safely reduce the number of maintained indices by eliminating from the graph each barrel with number of files less than a configured threshold. Then, they replicate the respective index across the minimal set of nodes that can search the files of the eliminated barrel.

A different study aims to improve metadata search efficiency by hierarchically partitioning the filesystem by access permissions [10]. This approach creates many small partitions, but the authors leave for future study the full merging of partitions with identical permissions. However, the above problem is essentially family clustering with  $L_s=100\%$  in the context of the present paper.

**Encrypted Storage** Song et al. describe techniques to securely search remote documents maintained in encrypted form [15]. The client queries the server through a key and a plaintext or encrypted keyword. The server identifies keyword locations through linear scan of the encrypted documents. For large datasets, the server may use inverted index of encrypted keywords, and encrypted or plaintext posting lists. In contrast, the Mafdet system inserts keyed hashes of document keywords into a Bloom filter at the server [1]. Thus, a client only submits keyword hashes to search for documents at the server.

Chang and Mitzenmacher use an encrypted bitmap to encode the presence of particular keywords in a document [6]. The user submits a permuted keyword identifier along with a key to search for the encrypted documents that contain the keyword. The only information leaked to the server is the keyword sharing among the documents. Instead, CryptDB supports keyword search over individually encrypted words of a text column in a relational database [12]. PRISM trans-

forms the problem of keyword search over encrypted files into privacy-preserving map and reduce tasks [4].

Pervez et al. assume that both files and inverted indices are stored in encrypted form at the cloud [11]. Authorized users submit encrypted search criteria to a third party, which homomorphically encrypts them before their transmission to the cloud server. The cloud server uses a user-specific key to re-encrypt the index for query evaluation.

**Online Social Networks** Keyword search in social networks is possible through a set of inverted indices with each index containing keyword occurrences (posting lists) of documents from particular users. Access control is enforced through intersection of the search result with the identifiers (author list) of documents authored by a particular set of users [2]. The authors examine alternative cost models to optimally include specific friends in the author list of each user, and introduce the HeapUnion operator to efficiently process multiple lists of document identifiers [3].

Hummingbird is a microblogging system that cryptographically hides from a user the topics on which other users follow her, and from third parties the fact that a user follows another user on a specific topic [8]. More generally, Cheng et al. enable fine-grain specification of access-control policies in user-to-user, user-to-resource and resource-to-resource relationships over social networks [7]. Hails provides data-flow confinement at the client and server side so that mutually-untrusted web applications can interact safely [9]. These are more general issues of access control in social networks, and lie beyond the scope of our present study.

## 6. CONCLUSIONS AND FUTURE WORK

We use clustering to identify documents with similar sets of authorized searchers. Accordingly, we generate shared indices for documents with common authorized searchers of sufficient volume. We experimentally show that with tunable parameters we achieve an indexing organization that combines low number of indices per user with low number of indices per document. In our future work we plan to integrate the Lethe workflow into a distributed search engine and experiment with a broad collection of datasets from collaborative environments, cloud storage and social networks.

## 7. ACKNOWLEDGEMENTS

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

## 8. REFERENCES

- [1] S. Artzi, A. Kieżun, C. Newport, and D. Schultz. Encrypted keyword search in a distributed storage system. Technical Report MIT-CSAIL-TR-2006-10, CSAIL, MIT, Feb. 2006.
- [2] T. A. Björklund, M. Götz, and J. Gehrke. Search in social networks with access control. In *Intl. Work. Keyword Search on Structured Data (KEYS)*, pages 4:1–4:6, Indianapolis, IN, June 2010.
- [3] T. A. Björklund, M. Götz, J. Gehrke, and N. Grimsmo. Workload-aware indexing for keyword search in social networks. In *ACM Intl. Conf.*

- Information and Knowledge Management (CIKM)*, pages 535–544, Glasgow, UK, Oct. 2011.
- [4] E.-O. Blass, R. D. Pietro, R. Molva, and M. Önen. PRISM - privacy-preserving search in MapReduce. In *Privacy Enhancing Technologies Symposium*, pages 180–200, Vigo, Spain, July 2012.
  - [5] S. Büttcher and C. L. A. Clarke. A security model for full-text file system search in multi-user environments. In *USENIX Conf. on File and Storage Technologies (FAST)*, pages 169–182, San Francisco, CA, Dec. 2005.
  - [6] Y.-C. Chang and M. Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *Intl. Conf. Applied Cryptography and Network Security*, pages 442–455, New York, NY, June 2005.
  - [7] Y. Cheng, J. Park, and R. Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *Intl. Conf. Social Computing/Intl. Conf. Privacy, Security, Risk and Trust (SocialCom/PASSAT)*, pages 646–655, Amsterdam, Netherlands, Sept. 2012.
  - [8] E. D. Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of Twitter. In *IEEE Symp. Security and Privacy*, pages 285–299, San Francisco, CA, May 2012.
  - [9] D. G. Giffin, A. Levy, D. Stefan, D. Terei, D. Mazières, J. C. Mitchell, and A. Russo. Hails: Protecting data privacy in untrusted web applications. In *USENIX Symp. Operating Systems Design and Implementation (OSDI)*, pages 47–60, Hollywood, CA, Oct. 2012.
  - [10] A. Parker-Wood, C. Strong, E. L. Miller, and D. D. Long. Security aware partitioning for efficient file system search. In *IEEE Symp. Massive Storage Systems and Technologies*, pages 1–14, Incline Village, NV, May 2010.
  - [11] Z. Pervez, A. A. Awan, A. M. Khattak, S. Lee, and E.-N. Huh. Privacy-aware searching with oblivious term matching for cloud storage. *Journal of Supercomputing*, 63(2):538–560, Feb. 2013.
  - [12] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *ACM Symp. Operating Systems Principles (SOSP)*, pages 85–100, Cascais, Portugal, Oct. 2011.
  - [13] A. Singh, M. Srivatsa, and L. Liu. Search-as-a-service: Outsourced search over outsourced storage. *ACM Transactions on the Web*, 3(4):13:1–13:33, Sept. 2009.
  - [14] D. K. Smetters and N. Good. How users use access control. In *Symp. On Usable Privacy and Security (SOUPS)*, Mountain View, CA, July 2009.
  - [15] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *IEEE Symp. Security and Privacy*, pages 44–55, Berkeley, CA, May 2000.
  - [16] P.-N. Tan, M. Steinbach, and V. Kumar. *Data Mining*, chapter 8. Addison-Wesley, May 2005.



# Community Detection in Anonymized Social Networks

Alina Campan

Department of Computer Science  
Northern Kentucky University  
Highland Heights, KY 41099, USA  
campana1@nku.edu

Yasmeen Alufaisan

Department of Computer Science  
The University of Texas at Dallas  
Richardson, TX 75080, USA  
yxa130630@utdallas.edu

Traian Marius Truta

Department of Computer Science  
Northern Kentucky University  
Highland Heights, KY 41099, USA  
trutat1@nku.edu

## ABSTRACT

Social media and social networks are embedded in our society to a point that could not have been imagined only ten years ago. Facebook, LinkedIn, and Twitter are already well known social networks that have a large audience in all age groups. Recently more trendy social sites such as Pinterest, Instagram, Vine, Tumblr, WhatsApp, and Snapchat are being preferred by the younger audience. The amount of data that those social sites gather from their users is continually increasing and this data is very valuable for marketing, research, and various other purposes. At the same time, this data usually contain a significant amount of sensitive information which should be protected against unauthorized disclosure. To protect the privacy of individuals, this data must be anonymized such that the risk of re-identification of specific individuals is very low. In this paper we study how well anonymized social networks preserve existing communities from the original social networks. To anonymize social networks we used two models, namely,  $k$ -anonymity for social networks and  $k$ -degree anonymity. To determine communities in social networks we used a community detection algorithm based on modularity quality function known as Louvain method. Our experiments on publically available datasets show that anonymized social networks satisfactorily preserve the community structure of their original networks.

## Categories and Subject Descriptors

H.2.7 [Database Management]: Database Administration – Security, integrity, and protection; K.4.1 [Computers and Society]: Public Policy Issues – Privacy; J.4 [Computer Applications]: Social and Behavioral Sciences – Sociology.

## General Terms

Algorithms, Experimentation, Human Factors.

## Keywords

Social Networks, Privacy, Anonymization, Community Detection, Modularity.

## 1. INTRODUCTION

Social media and social networks are embedded in our society to a point that could not have been imagined only ten years ago. Facebook, LinkedIn, and Twitter are already well known social networks that have a large audience in all age groups. Recently

more trendy social sites such as Pinterest, Instagram, Vine, Tumblr, WhatsApp, and Snapchat are being preferred by the younger audience [26]. The amount of data that those social sites gather from their users is continually increasing and this data is very valuable for marketing, research, and various other purposes. At the same time, this data usually contain a significant amount of sensitive information which should be protected against unauthorized disclosure. The above social sites treat seriously the privacy of their members and they provide a series of privacy controls and a privacy policy regarding of how the collected data is used. First, the privacy controls allow individuals to set up their privacy preferences/settings. Using these settings, a user may choose what personal information is available to each group of friends or what personal information is available to everyone on the internet. Second, the privacy policy lists how the social site will use the data from their users and how this data can be shared with third party companies such as advertising companies, etc. To protect the privacy of individuals, this data must be anonymized such that the risk of re-identification of specific individuals is very low.

In this paper we focus only on social network data model, which is one of the most common data models used in social media. The social network data (also referred as graph data or simply network data) should be made anonymous before being released in order to protect the privacy of individuals that are included in this social network. Due to a wide variety of problem assumptions, a standard social network anonymization model does not exist. One important assumption is what constitutes sensitive information which needs to be protected against disclosure. In general, either identity of individuals, their relationship, and/or part of their social network node content is considered sensitive [18]. A second aspect of anonymization is what anonymization approach is more appropriate to follow, and there are three choices that are analyzed in the literature: anonymization via clustering, anonymization via graph modification, and a hybrid approach [3, 7, 37, 39]. Considering these choices, it is not a surprise that the resulting anonymized networks are very dissimilar in terms of structure and in terms of preserving the original graph properties. In this paper we consider only the identity of individuals being sensitive information and we analyze two anonymization models. These models are:  $k$ -anonymity for social networks [7], a model from the anonymization by clustering family, which can be enforced on a network by using the *Sangre* algorithm, and  $k$ -degree anonymity [18], a graph modification approach, enforced by the *Fast K-Degree Anonymization (FKDA)* algorithm [19].

The purpose of this work is to study how well anonymized social network preserve existing communities from the original social networks. Communities (also known as clusters) are groups of nodes from a social network which likely have similar properties or characteristics [12] Community detection is well studied in the literature and many different community detection algorithms

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

7th International Workshop on Privacy and Anonymity in the Information Society (PAIS'14) March 28, 2014, Athens, Greece

have been presented in social network analysis literature. A good survey of these algorithms can be found in [12]. For this paper we focus on a specific community detection method known as Louvain method [4, 27], which is a heuristic algorithm based on modularity optimization [23]. The modularity is a quality function that can be computed for a graph partitioned in communities. Modularity has received a wide attention in recent years being used as a quality function in many community detection algorithms, to assess the stability of partitions [21], in determining graph visualization layouts [24], and in graph summarization [2].

To study how well communities are preserved in anonymized social networks we follow several steps. First, we anonymize several real social networks using *Sangreea* and *Fast K-Degree Anonymization* algorithms. Second, we de-anonymize networks masked with *Sangreea* to allow fair comparison between the original and the anonymized network (details will be provided later). And third we use Louvain community detection algorithm to compare how well the communities are preserved between the original networks and their anonymized (via *Fast K-Degree Anonymization*) and de-anonymized *Sangreea* versions.

The remaining of this paper is structured as follows. Section 2 presents related work. Section 3 describes the anonymity models used in this paper. Section 4 presents the de-anonymization models that we used with the anonymization via clustering networks. Section 5 describes the modularity function, the community detection algorithm used in this paper, and how we compute the community preservation. Section 6 contains the experimental results. Section 7 summarizes our conclusions.

## 2. RELATED WORK

This paper applies several new findings in data privacy, social network analysis, and graph generators in a new more practical problem. To our knowledge this is the first paper that addresses how well the existing communities in social networks are preserved when these social networks are anonymized.

Related to this work are a series of papers that analyses the usefulness of anonymized social network for other social analysis tasks. Most of the previous works compare how well structural properties (diameter [14], centrality measures [13], clustering coefficients [33, 34] and/or topological indices [20]) are preserved between the original social networks and their anonymized versions. Three such papers considers anonymization via clustering in their analysis and they differs in which structural property are analyzed and how the anonymization/de-anonymization is performed [1, 31, 32]. Other papers that discuss structural property preservation focus on how specific graph modification approaches ( $k$ -automorphism [29],  $k$ -isomorphism [11], and  $k$ -symmetry [35]) preserve a subset of those structural properties. In other related work, comparison of the most influential nodes and the spread of influence in social networks were performed between the original social networks and the anonymized/de-anonymized networks [8].

As already mentioned, related to this work are social network anonymization models, community detection in social networks, and graph generators models. Each of these topics is well covered in research literature. A good survey of existing social network anonymization models as well as other issues regarding privacy in social networks is covered in [38]. Various community detection techniques are also well studied in the literature [12, 17]. A survey of graph generators models is presented in [10]. In this paper we use the *Erdos-Renyi* random network model [5] and *R-MAT* power law model [9].

## 3. SOCIAL NETWORK ANONYMITY MODELS

In this section the two anonymity models used in this paper,  $k$ -anonymity for social networks and  $k$ -degree anonymity, are briefly introduced. Since in this paper our focus is on community preservation based on the social networks structure, we make the additional simplifying assumption that the nodes in the social network do not have quasi-identifier attributes (such as *Age* and *ZipCode*); accordingly, the anonymization process is based on the social network structure only. Sensitive attribute values that need to be protected from potential intruders (such as *ICD9Code* and *Income*) are preserved in the social network.

Consider an initial social network modeled as a simple undirected graph  $G = (\mathcal{N}, \mathcal{E})$ , where  $\mathcal{N}$  is the set of nodes and  $\mathcal{E}$  is the set of edges. Only binary relationships are allowed in this model. Additionally, all relationships are of the same type and they are represented as unlabeled undirected edges. These edges are assumed to be known by an intruder. Based on this graph structure, an intruder is able to identify individuals and to reveal their sensitive information due to the uniqueness of their neighborhoods.

We illustrate an example of social network, labeled  $G_1$ , in Figure 1. This network has 12 nodes and 12 edges.

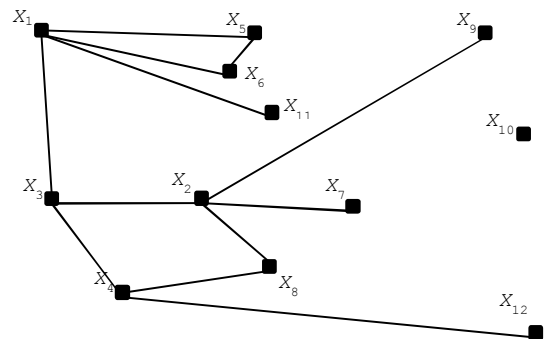


Figure 1. Social network example,  $G_1$ .

### 3.1 K-Anonymity for Social Networks

In this model, the nodes from the social network are partitioned into pairwise disjoint clusters based on a similarity criteria. These clusters are generalized to super-nodes, which may be connected by super-edges. The goal of this process is to make any two nodes belonging to the the same cluster indistinguishable based on their relationships. To achieve this objective, Campan and Truta developed intra-cluster and inter-cluster edge generalization techniques that were used for creating super-nodes and super-edges [7]. To satisfy the  $k$ -anonymity for social networks clustered model – model derived from the well-known  $k$ -anonymity property for microdata [28, 30], each cluster must have at least  $k$  nodes. The algorithm used to create these clusters is named Social Network Greedy Anonymization (*Sangreea*). This algorithm partitions the set of nodes in the social network into a set of disjoint clusters with size at least  $k$  and with nodes as similar to each other as possible in terms of their neighborhoods.

In the anonymized network, each cluster is replaced by a super-node and edges from the original network are generalized via an edge generalization process which preserves the number of edges, in other words, it does not add or delete edges. The edge

generalization process is divided into two components: edge intra-cluster generalization and edge inter-cluster generalization.

Edge intra-cluster generalization is a process in which each of the clusters is generalized into a single super-node and the information released with it is the pair of values  $(|cl|, |\mathcal{E}_{cl}|)$ , where  $|X|$  represents the cardinality of the set  $X$ ,  $cl$  represents the set of nodes in the cluster, and  $\mathcal{E}_{cl}$  represents the set of edges that connect two nodes from  $cl$ . An example of such super-node information would be  $(4, 3)$ , which means that the cluster has four of the original nodes with three edge between them. Hiding the precise connectivity information between nodes in the same cluster will protect the identity of cluster's nodes.

Edge inter-cluster generalization is a similar process for edges between two clusters. In the anonymized graph, the set of inter-cluster edges between any two clusters is generalized into one single super-edge. The information released due to this process is the value  $|\mathcal{E}_{cl_1, cl_2}|$ , where  $cl_1$  and  $cl_2$  are the two clusters and  $\mathcal{E}_{cl_1, cl_2}$  represents the set of edges that connect the two clusters. In other words, each super-edge is described by the number of edges connecting nodes within the two super-nodes. The time complexity of Sangreeta is  $O(n^2)$ . For complete details of the Sangreeta algorithm please consult [7].

Figure 2 shows the anonymized network,  $\mathcal{AG}_1$  that was obtained by applying Sangreeta algorithm to the social network  $G_1$  (see Figure 1). This anonymized network satisfies 4-anonymity for social network property ( $k = 4$ ).

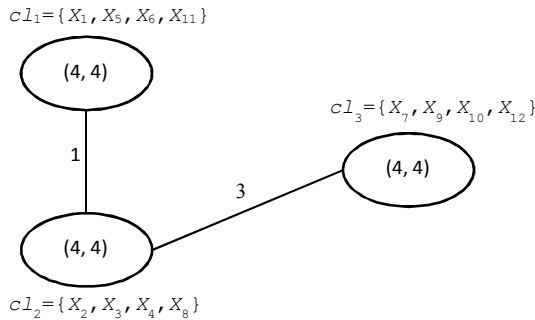


Figure 2. Anonymized social network,  $\mathcal{AG}_1$ .

### 3.2 K-Degree Anonymity

$K$ -degree anonymity protects against intruders' attacks with background knowledge that is limited to nodes' degree. A social network is  $k$ -degree anonymous if for every node  $X$  in the network, there are at least  $k-1$  other nodes with the same degree as the node  $X$  [18]. While an initial algorithm to create a  $k$ -degree anonymous network was proposed in [18], we used for this paper the *Fast K-Degree Anonymization (FKDA)* algorithm proposed by Lu et al. [19].

FKDA anonymizes a social network by adding edges in a greedy fashion until the network is  $k$ -degree anonymous. First, the nodes of the original graph are separated into several groups. Second, each predetermined group will be anonymized by adding edges to the nodes in the group until all the nodes in the group have the same degree. If anonymization cannot be achieved for a group in this edge creation algorithm, a more relaxed approach of adding edges is allowed, where nodes in the group being anonymized are connected to any nodes in the graph. The performing of the

relaxed addition can destroy the anonymity of nodes processed in previous steps – and if this happens, the whole process is restarted from scratch. The time complexity for FKDA is  $O(n^2)$  in the worst case, where  $n$  is the total number of nodes in the network. For complete details of the FKDA algorithm please consult [19].

Figure 3 illustrate the anonymized network,  $\mathcal{AG}_2$  that was obtained by applying FKDA algorithm to the social network  $G_1$  (see Figure 1). The dashed lines represent the new relationships added by FDKA algorithm. In this anonymized network the nodes  $X_1, X_2, X_3,$  and  $X_4$  have degree 4; the nodes  $X_5, X_6, X_7,$  and  $X_8$  have degree 2, and the remaining nodes  $X_9, X_{10}, X_{11},$  and  $X_{12}$  have degree 1. This network satisfies 4-degree anonymity ( $k = 4$ ).

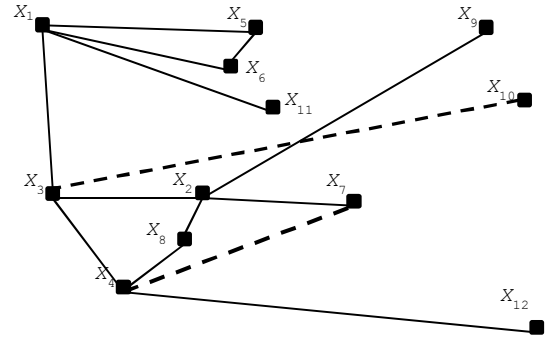


Figure 3. Anonymized social network,  $\mathcal{AG}_2$ .

### 4. DE-ANONYMIZATION PROCESS

To compare communities between social networks and  $k$ -degree anonymous social network is easier since both the initial and anonymized networks have the same number of nodes and only the number of edges differ (see Figure 1 and Figure 3). This comparison is more difficult in case of  $k$ -anonymous social networks because the number of nodes in the anonymized network is reduced by a factor of  $k$  from the initial social network. To avoid this problem we “de-anonymize”  $k$ -anonymous social networks using two different models to try to revert the anonymization process and create replicas of the original network. The de-anonymized networks will have the same number of nodes and edges as the original network, allowing therefore for a fair comparison of communities.

Two possible de-anonymized social networks of the anonymized network  $\mathcal{AG}_1$  (see Figure 2), labeled  $\mathcal{DG}_1$  and  $\mathcal{DG}_2$ , are shown in Figures 4 and 5. Notice that they have the same number of nodes and edges as the initial social network  $G_1$ , but they have a different structure.

To de-anonymize a  $k$ -anonymous social network we re-use the two methods presented in [1, 32], *Uniform De-anonymization* [32] and *R-MAT De-anonymization* [1]. Uniform De-anonymization will randomly create edges between nodes within each super-node up to the number of edges in that super-node, and between nodes from different super-nodes until the number of generated edges corresponds with the super-edge weight (similar with Erdos-Renyi random graph generator method). The R-MAT De-anonymization method is based on the assumption that many real-world networks are scale-free, and their nodes degree distribution follows a power-law. A complete description of this de-anonymization method can be found in [1].



## 5.1 Community Preservation

Using Louvain method we can compute communities for the initial social networks, the  $k$ -degree anonymous social networks (Section 3.2), and the de-anonymized  $k$ -anonymous social networks (Sections 3.1 and 4). To compare the results between an anonymized social network and the corresponding initial social network we simply count how many nodes from the original communities remained in the same community after the processes of anonymization and de-anonymization. We illustrate this approach with the following example. Figure 6 shows the initial social network, labeled  $\mathcal{SN}_1$ . Figure 7 shows a social network that was obtained from the initial social network by applying SangreA algorithm with  $k = 2$  and then the R-MAT de-anonymization procedure ( $\mathcal{SN}_2$ ). Figure 8 shows a 2-degree anonymous social network obtained by applying FKDA algorithm with  $k = 2$  ( $\mathcal{SN}_3$ ).

Table 1 shows the communities and how they are preserved between  $\mathcal{SN}_1$  and  $\mathcal{SN}_2$ , in other words for  $k$ -anonymity for social networks privacy model. Table 2 illustrate the communities and how they are preserved between  $\mathcal{SN}_1$  and  $\mathcal{SN}_3$ , in other words for  $k$ -degree anonymity privacy model. The communities were obtained using Louvain method.

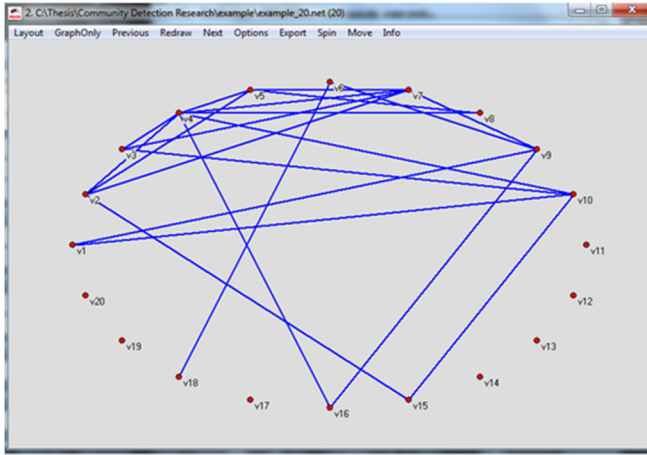


Figure 6. Initial social network,  $\mathcal{SN}_1$ .

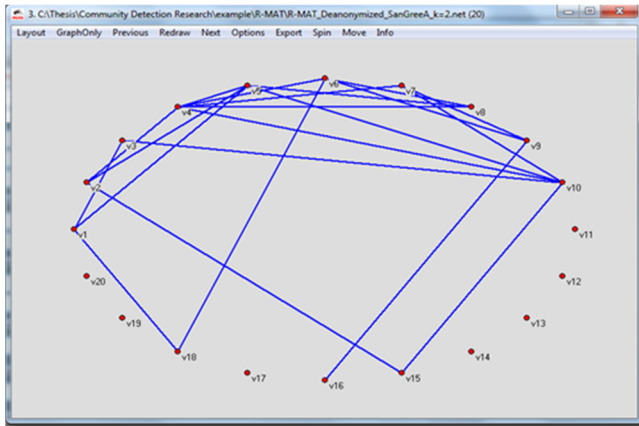


Figure 7. De-anonymized social network,  $\mathcal{SN}_2$ .

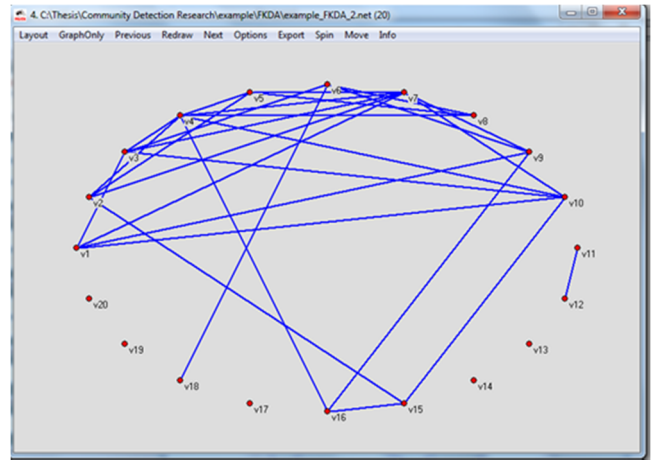


Figure 8. 2-degree anonymous social network,  $\mathcal{SN}_3$ .

To compute the % preservation column, for each community from  $\mathcal{SN}_1$  we select a corresponding community from  $\mathcal{SN}_2$  or  $\mathcal{SN}_3$  that contain the maximum number of elements from the initial community. For instance for the third community from Table 2,  $\{6, 9, 16, 18\}$ , the best match is the community  $\{6, 9, 18\}$  and the % preservation is  $3/4$ . To find out an overall community preservation measure we average the results from the % preservation column and we obtain the following results:

- $CommunityPreservation(\mathcal{SN}_1, \mathcal{SN}_2) = 89\%$
- $CommunityPreservation(\mathcal{SN}_1, \mathcal{SN}_3) = 93\%$ .

Table 1. Community preservation –  $k$ -anonymity for social networks

Community ID	Communities in $\mathcal{SN}_1$	Communities in $\mathcal{SN}_2$	% Preservation
1	1, 3, 10, 15	1, 2, 3, 5, 10, 15	100%
2	2, 4, 5, 7, 8	4, 6, 8, 18	40%
3	6, 9, 16, 18	7, 9, 16	50%
4	11	11	100%
5	12	12	100%
6	13	13	100%
7	14	14	100%
8	17	17	100%
9	19	19	100%
10	20	20	100%

Table 2. Community preservation –  $k$ -degree anonymity

Community ID	Communities in $\mathcal{SN}_1$	Communities in $\mathcal{SN}_3$	% Preservation
1	1, 3, 10, 15	1, 3, 7, 10	75%
2	2, 4, 5, 7, 8	2, 4, 5, 8, 15, 16	80%
3	6, 9, 16, 18	6, 9, 18	75%
4	11	11, 12	100%
5	12	13	100%
6	13	14	100%
7	14	17	100%
8	17	19	100%
9	19	20	100%
10	20	-	100%

## 6. EXPERIMENTS AND RESULTS

We study the preservation of communities between original and anonymized/de-anonymized versions of the following publically available datasets:

- **Cond** is a collaboration network of scientists [22]. This network is undirected and consists of 16,726 nodes, 47,594 edges, and 1247 communities. The number of communities is obtained using Louvain method from Pajek network analysis tool. Two scientists are considered connected (have an edge between them) if they coauthored a paper.
- **Enron** dataset is a network of email exchanges [15, 16]. It is an undirected network with 36,692 nodes, 183,831 edges, and 1286 communities. Each node in this network represents an email address. An edge exists between two nodes if at least one email was sent from one node to the other from that edge.
- **YouTube** dataset is an undirected social network [36]. The network has 1,157,827 nodes and 2,987,624 edges. Due to the large number of nodes and edges in the network, we extracted three sub-graphs from it. Each sub-graph is a well-defined community from the original network. Again, we used Louvain method from Pajek to extract the communities. YouTube network has 30,814 communities. Only six of these communities have number of nodes in the range between 15,000 and 40,000 which is the range of nodes we look for in our experiments. We will refer to these communities as the preferred-communities. When creating a sub-graph for a community, we retained only the nodes that members of the specified community and the edges that connect these selected nodes.

After creating the sub-graphs for the preferred-communities, we chose three sub-graphs as our initial social networks based on a unique feature for each one of them. Following is the description of these networks:

- **YouTubeLargest** is the largest community in YouTube preferred-communities. It has 37,530 nodes, 121,337 edges, and 363 communities. We used the number of nodes to measure the size of the communities and determine the largest one.
- **YouTubeCompact** is the most compact community from YouTube preferred-communities. We used the Clustering Coefficient to measure the compactness of the network. When using Pajek to measure the Clustering Coefficient [33, 34] for YouTubeCompact, Watts-Strogatz Clustering Coefficient was 0.24883441 and Network Clustering Coefficient (Transitivity) was 0.04206904, which are the largest values among the other communities in the preferred-communities. YouTubeCompact contains 20,272 nodes, 28,026 edges, and 128 communities.
- **YouTubeRandom** is a community that was chosen randomly from YouTube network preferred-communities. It has 22,409 nodes, 27,927 edges and 143 communities.

The steps for the experiments to measure the community preservation are:

- First, we started with the initial networks (**Cond**, **Enron**, **YouTubeLargest**, **YouTubeCompact**, **YouTubeRandom**) described previously. We anonymized these networks with

FKDA and Sangreeta using several anonymity parameter  $k$ : 5, 10, 15, 20, 25, and 50.

- For each  $k$ -anonymous social network we generated 5 de-anonymized networks using Uniform De-anonymization and 5 de-anonymized networks using R-MAT De-anonymization (Section 4). Repeating the de-anonymization process 5 times was done because of the randomness of the de-anonymization process. In this step, we also run the de-anonymization processes for a  $k$ -anonymous social network with  $k = n$  (size of the network), this is equivalent with executing Uniform and R-MAT de-anonymization without having any knowledge regarding the initial network structure except its size (the number of nodes and the number of edges).
- After that, we extracted the communities of the original networks using Louvain community detection method in Pajek using the following steps: Network-> create partition->Communities->Louvain Method-> Multi- Level Coarsening + Multi- Level Refinement.
- Then, we extracted the communities from  $k$ -degree anonymous networks and the de-anonymized networks as described in the previous step.
- To compute the community preservation, we mapped every community detected in the original network to the best match community in the anonymized/de-anonymized networks. A best match community would be a community that has the most nodes from the original community. After that, we compute the percentage of nodes that remain the same community before and after the anonymization/de-anonymization process. Finally, we take the average community preservation for all the communities in the original network. An example of this process is shown in Section 5.
- Since we generated 5 de-anonymized networks for each  $k$ -anonymous social network, the community preservation determined in those cases is averaged.

The workflow of our experiments is shown in Figure 9. The average community preservation (% preservation) results for the community preservation experiments are shown in Figures 10-14 for Cond, Enron, YouTubeLargest, YouTubeCompact, and YouTubeRandom datasets. The vertical axis represents the percentage of the average community preservation for the networks. The last  $k$  value represents the size of the network and we report in this case the community preservation when there is no  $k$ -anonymous social network available; in other words all the nodes and edges are collapsed into a super-node where the number of nodes and the number of edges for the entire initial network are reported. The community preservation for this case represents the baseline value, and in all experiments the community preservation is superior to this baseline case.

For Cond network (Figure 10), FKDA had a good preservation of the communities of the original network and there were a noticeable decrease only in the case where  $k = 50$ . On the other hand, R-MAT and Uniform de-anonymization had almost identical preservation for the communities of the original network except for the case where  $k = 5$ , R-MAT had much better preservation than Uniform.

For Enron network (Figure 11), FKDA preserved the communities of the original network very well. R-MAT de-anonymization

preserved the communities of the original networks well when  $k$  was small and the community preservation started to drop rapidly as  $k$  got larger. Uniform de-anonymization had the lowest preservation of communities when  $k$  was 5 and 10, but for the larger values of  $k$ , Uniform performed slightly better than R-MAT.

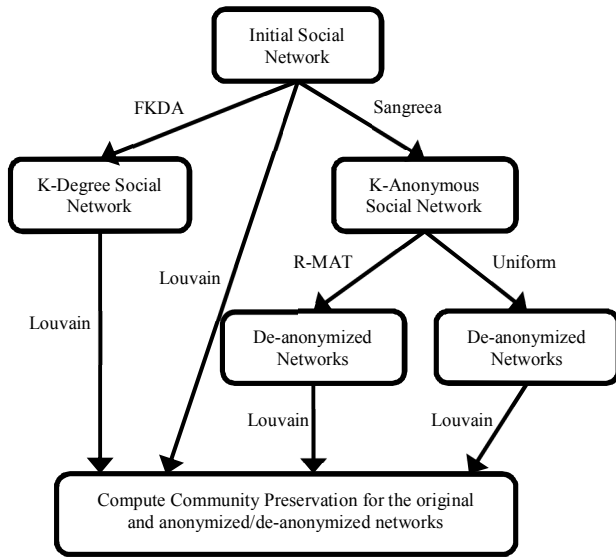


Figure 9. Workflow for community preservation experiments.

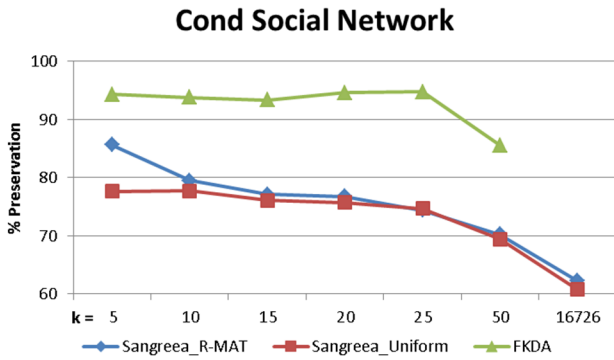


Figure 10. % preservation for Cond.

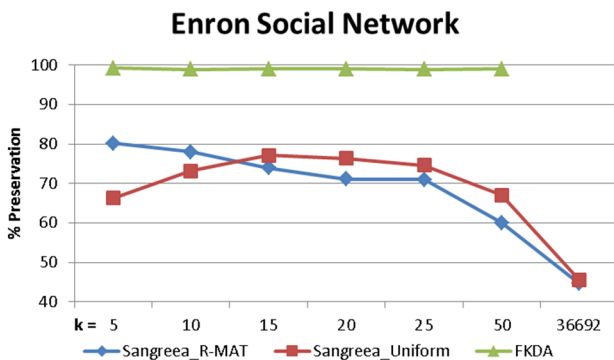


Figure 11. % preservation for Enron.

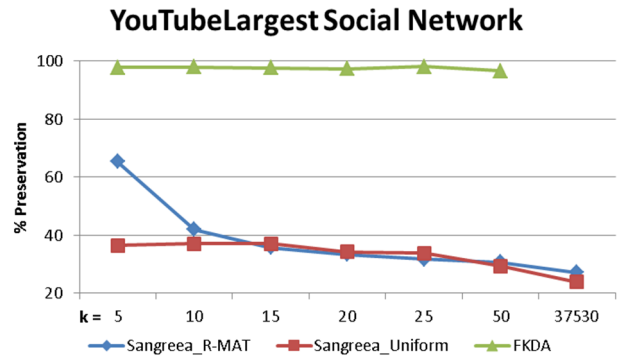


Figure 12. % preservation for YouTubeLargest.

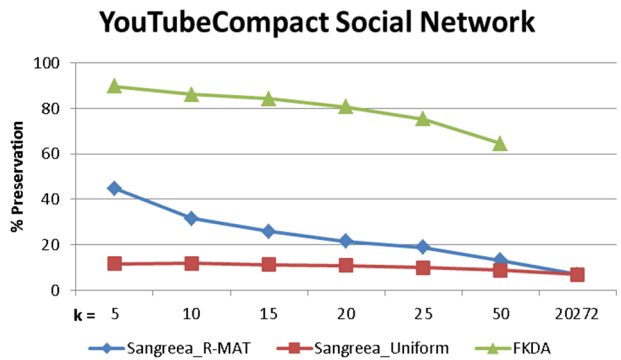


Figure 13. % preservation for YouTubeCompact.

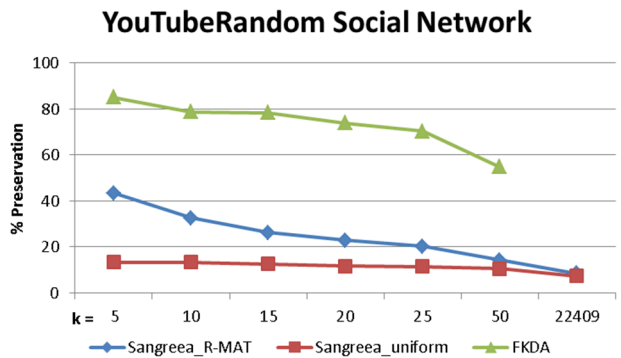


Figure 14. % preservation for YouTubeRandom.

FKDA also preserved the communities well for YouTubeLargest network for all  $k$  values (Figure 12). And as with Cond network, R-MAT performed better when  $k$  was 5 but for the larger values of  $k$  R-MAT and Uniform had almost the same preservation.

For YouTubeCompact (Figure 13) and YouTubeRandom (Figure 14) we had similar curves for FKDA, R-MAT De-anonymization, and Uniform De-anonymization. FKDA had the best preservation of communities followed by R-MAT De-anonymization. For both of these cases the preservation of communities decreased continuously. However, Uniform De-anonymization had the worst community preservation with an almost steady line.



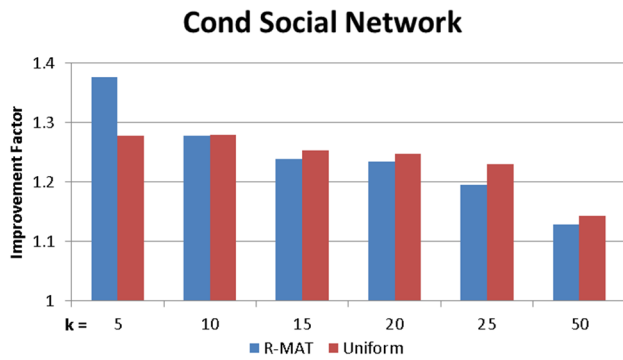


Figure 15. Improvement factor for Cond.

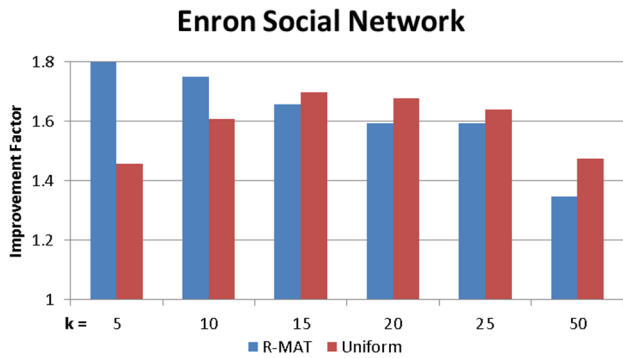


Figure 16. Improvement factor for Enron.

Based on the results reported in Figures 10-14, we conclude that FKDA algorithm preserves very well the community structure of the initial social network. This result is expected since  $k$ -degree anonymity keeps most of the initial structure of the social network. However, as pointed out in Section 3,  $k$ -degree anonymity is a “weak” anonymity model since it assumes that an intruder has only knowledge about the degree of individuals in the network and not about the network structure. The other two methods used in conjunction with  $k$ -anonymity for social network model (Uniform and R-MAT de-anonymization) while clearly outperformed by FKDA, also preserves to some extent the community structure of the original network. As expected R-MAT de-anonymization is, in general, outperforming Uniform de-anonymization. Figures 15-19 show the improvement factor of those two methods compared with the communities that exist in a random graph (uniform random graph and R-MAT random graph) with the same number of node and vertices (the improvement factor for this baseline case is 1).

As expected, the smaller the value of  $k$ , the communities are better preserved. However, this is not true for some of the experiments. For FKDA, since the results are very similar for all values of  $k$ , in some cases the % preservation increases when  $k$  increases. This is due to addition of edges within original communities for larger  $k$  which contribute to their preservation in the anonymized dataset. For de-anonymization the only such inversion is detected for Enron dataset and Uniform de-anonymization method. This is likely because the Sangreia algorithm breaks larger communities in super-nodes of size  $k$ , and then the Uniform de-anonymization will generate edges between vertices from different communities

such that the initial communities cannot be found in the final de-anonymized networks. R-MAT de-anonymization is able to better preserve such community due to its edge generation procedure that follows better the degree distribution of the initial network.

It is also worth noting that in all three experiments that use YouTube dataset, the communities are well preserved in case of R-MAT de-anonymization and low  $k$  values, in particular for YouTubeCompact and YouTubeRandom, the improvement factor is over 5 (for  $k = 5$ ). This is due to a combination of factors. First, as stated above, the R-MAT de-anonymization is preserving the original network structure better. And second, the communities are not well preserved in case of a random graph, thus the % preservation is very low for the baseline case.

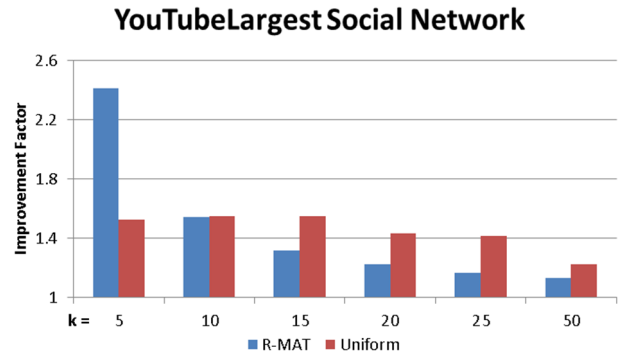


Figure 17. Improvement factor for YouTubeLargest.

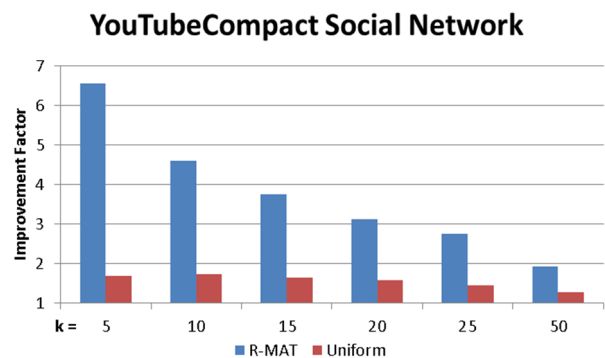


Figure 18. Improvement factor for YouTubeCompact.

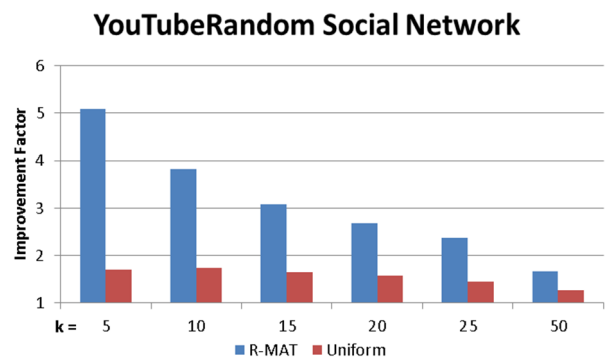


Figure 19. Improvement factor for YouTubeRandom.

## 7. CONCLUSIONS AND FUTURE WORK

In this paper, we studied how well communities are preserved when social networks are anonymized. We analyzed two models  $k$ -anonymity for social networks and  $k$ -degree anonymity. Our results show that FKDA algorithm used to create a  $k$ -degree anonymous network preserved very well the communities from the initial networks. The de-anonymization methods used after the social networks were anonymized with Sangreeta algorithm (to become  $k$ -anonymous social networks) also are able to preserve, although less successfully than FKDA, the initial communities. In most experiments the R-MAT de-anonymization outperforms the Uniform de-anonymization.

From the privacy point of view,  $k$ -anonymity for social networks enforces a much stronger model than  $k$ -degree anonymity.  $K$ -degree anonymity only considers the degree of each node as possible background knowledge for an intruder; so an intruder with more knowledge about the network structure can breach the privacy of a  $k$ -degree anonymous network. For  $k$ -anonymous networks, an intruder with any background knowledge about the structure of the network cannot breach the privacy of the network.

There are several future research directions that we want to pursue. First, the community preservation measure is useful when the number of communities is roughly the same between the initial and anonymized social network. When the number of communities in the anonymized social network decreases it is likely that the original communities are preserved in larger communities. Our measure does not distinguish between these two situations and, therefore, we intend to create a more robust way of comparing communities' preservation. Second, the criterion to construct super-nodes in Sangreeta is based on neighbor similarities between all nodes from the network. We intend to adapt Sangreeta algorithm to create super-nodes with nodes that belong to one community, and in this way we hope to increase the community preservation.

## 8. REFERENCES

- [1] Alufaisan Y. and Campan A. 2013. Preservation of centrality measures in anonymized social networks. *Proceedings of the ASE/IEEE International Conference on Privacy, Security, Risk, and Trust (PASSAT 2013)*, Washington D.C., USA.
- [2] Arenas A., Duch J., Fernandez A., Gomez S. 2007. Size reduction of complex networks preserving modularity. *New J. Phys.* 9, art. no. 176.
- [3] Bhagat S., Cormode G., Krishnamurthy B., and Srivastava D. 2009. Class-based graph anonymization for social network data. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*.
- [4] Blondel V. D., Guillaume J.-L., Lambiotte R., Lefebvre E. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 1742-5468.
- [5] Bollobás B. 2001. Random graphs, 2nd ed., *Cambridge University Press*.
- [6] Brandes U., Delling D., Gaertler M., Gorke R., Hoefer M., Nikoloski Z., and Wagner D. 2008. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20, No. 2, February 2008, 172-188.
- [7] Campan A. and Truta T. M. 2008. A clustering approach for data and structural anonymity in social networks. *Proceedings of the 2nd ACM SIGKDD International Workshop on Privacy, Security, and Trust in KDD (PinKDD)*.
- [8] Campan A. and Alufaisan Y. 2013. Social network anonymization and influence preservation. *Proceedings of the International Conference on Data Mining (DMIN'13)*, Las Vegas, Nevada, USA.
- [9] Chakrabarti D., Zhan Y., and Faloutsos C. 2004. R-MAT: A recursive model for graph mining. *Proceedings of the SIAM International Conference on Data Mining (SDM'04)*, 442-446.
- [10] Chakrabarti D. and Faloutsos C. 2006. *Graph mining: laws, generators, and algorithms*. ACM Computing Surveys, Volume 38, Article 2.
- [11] Cheng J., Fu A. W. C., and Liu J. 2010. K-isomorphism: privacy preserving network publication against structural attacks. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 459-470, DOI= <http://doi.acm.org/10.1145/1807167.1807218>.
- [12] Fortunato S. 2010. Community detection in graphs. *Physics Reports*, Volume 486, Issues 3-5, 75-174, DOI=<http://dx.doi.org/10.1016/j.physrep.2009.11.002>.
- [13] Freeman L. C. 1979. Centrality in social networks: conceptual clarification. *Social Networks*, vol. 1, no. 3, 215-239.
- [14] Harary F. 1994. Graph theory. *Addison-Wesley*.
- [15] Klimmt B. and Yang Y. 2004. Introducing the Enron corpus. *CEAS conference*.
- [16] Leskovec J., Lang K., Dasgupta A., and Mahoney M. 2009. Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, Vol. 6, No 1, 29-123.
- [17] Leskovec J., Lang K.L., and Mahoney M.W. 2010. Empirical Comparison of Algorithms for Network Community Detection. *Proceedings of the World Wide Web Conference (WWW 2010)*, Raleigh, North Carolina USA, 631-640.
- [18] Liu K. and Terzi E. 2008. Towards identity anonymization on graphs. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 93-106, DOI= <http://doi.acm.org/10.1145/1376616.1376629>.
- [19] Lu X., Song, Y., and Bressan S. 2012. Fast identity anonymization on graphs. *Proceedings of the 23rd International Conference on Database and Expert Systems Applications (DEXA)*, 281-295.
- [20] Lukovits I., Nikolic S., and Trinajstic N. 2002. On relationships between vertex-degrees, path-numbers and graph valence-shells in trees. *Chemical Physics Letter*, Vol. 354, 417-422.
- [21] Massen C. P., Doye, J. P. K. 2006. Thermodynamics of community structure. ePrint [arXiv:cond-mat/0610077](http://arxiv.org/abs/cond-mat/0610077).
- [22] Newman, M. E. J. 2001. The structure of scientific collaboration networks. *Proc. Natl. Acad. Sci. USA* 98, 404-409.
- [23] Newman M. E. J., Girvan M. 2004. Finding and evaluating community structure in networks. *Physical Review*, E 69 (2), 026113.
- [24] Noack A. 2009. Modularity clustering is force-directed layout. *Physical Review*, E 79 (2), 026102.

- [25] Nooy W., Mrvar A., and Batagelj V. 2011. Exploratory social network analysis with pajek. *Revised and Expanded Second Edition, Structural Analysis in the Social Sciences*, Vol. 34, Cambridge University Press, 2011.
- [26] Olson, P. 2013. Teenagers say goodbye to Facebook and hello to messenger apps. *The Observer Journal*, Saturday 9 November 2013, Online at: <http://www.theguardian.com/technology/2013/nov/10/teenagers-messenger-apps-facebook-exodus>
- [27] Rotta R., Noack A. 2011. Multilevel local search algorithms for modularity clustering. *Journal of Experimental Algorithms*, Volume 16, Article no 2.3, DOI=<http://doi.acm.org/10.1145/1963190.1970376>.
- [28] Samarati P. 2001. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 6, 1010-1027.
- [29] Song Y., Nobari S., Lu X., Karras P., and Bressan S. 2011. On the privacy and utility of anonymized social networks. *Proceedings of the iiWAS'11*, Ho Chi Minh City, Vietnam.
- [30] Sweeney L. 2002. K-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness, and Knowledge-based Systems*, vol. 10, no. 5, 557 – 570.
- [31] Truta T.M., Campan A., Gasmi A., Cooper N., and Elstun A. 2011. Centrality preservation in anonymized social networks. *Proceedings of the International Conference on Data Mining (DMIN'11)*, Las Vegas, Nevada, USA.
- [32] Truta T.M., Campan A., and Ralescu A.L. 2012. Preservation of structural properties in anonymized social networks. *Proceedings of the Collaborative Communities for Social Computing Workshop (CCSocialComp-2012)*, held in conjunction with CollaborateCom-2012, Pittsburgh, Pennsylvania, USA.
- [33] Wasserman S. and Faust K. 1994. Social network analysis: methods and applications. *Cambridge: Cambridge University Press*.
- [34] Watts D. J. and Strogatz S. H. 1998. Collective dynamics of 'small-world' networks. *Nature*, Vol. 393, 440-442.
- [35] Wu W., Xiao Y., Wang W., He Z., and Wang Z. 2010. K-symmetry model for identity anonymization in social networks. *Proceedings of the Extending Database Technology Conference (EDBT)*, 111-122, DOI=<http://doi.acm.org/10.1145/1739041.1739058>.
- [36] Yang J. and Leskovec J. 2012. Defining and evaluating network communities based on ground-truth. *Proceedings of the International Conference on Data Mining (ICDM)*.
- [37] Zheleva E. and Getoor L. 2007. Preserving the privacy of sensitive relationships in graph data. *Proceedings of the ACM SIGKDD Workshop on Privacy, Security, and Trust in KDD (PinKDD)*, 153-171.
- [38] Zheleva E., Terzi E., and Getoor L. 2012. Privacy in social networks. *Synthesis Lectures on Data Mining Series*. Book published by Morgan and Claypool Publishers.
- [39] Zhou B. and Pei J. 2008. Preserving privacy in social networks against neighborhood attacks. *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 506-515.

# Secure Multi-Party linear Regression

Fida Dankar  
University of Ottawa  
IBM Canada  
fidamark@ca.ibm.com

Renaud Brien  
University of Ottawa  
Rbrie047@uottawa.ca

Carlisle Adams  
University of Ottawa  
cadams@eecs.uottawa.  
ca

Stan Matwin  
Dalhousie University  
stan@cs.dal.ca

## ABSTRACT

Increasing efficiency in hospitals is of particular importance. Studies that combine data from multiple hospitals/data holders can tremendously improve the statistical outcome and aid in identifying efficiency markers. However, combining data from multiple sources for analysis poses privacy risks. A number of protocols have been proposed in the literature to address the privacy concerns; however they do not fully deliver on either privacy or complexity. In this paper, we present a privacy preserving linear regression model for the analysis of data coming from several sources. The protocol uses a semi-trusted third party and delivers on privacy and complexity.

## Categories and Subject Descriptors

D.3.3 [Computers and Society]: Public Policy Issues– *Privacy*

H.2.8 [Database Management]: Database Applications– *Data Mining*.

## General Terms

Algorithms, Security, Theory.

## Keywords

Linear regression, privacy preserving data mining, secure multiparty computation.

## 1. INTRODUCTION

Hospitals are under a lot of pressure to increase their efficiency. They need to see more patients and reduce their costs without increasing resources [1]. Increasing the efficiency of critical resources, such as surgeons and operating rooms, while keeping the same quality of care is of particular focus [1]. For example, surgery completion time is an indicator of critical resources efficiency. Several studies have presented interesting explanation for the variation in surgery completion times [2]–[4], among the culprits are individual, team and organizational experience, learning curve heterogeneity and workload. These studies however work with raw data coming from a single source.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

7th International Workshop on Privacy and Anonymity in the Information Society (PAIS'14) March 28, 2014, Athens, Greece

Combining data from multiple sources or hospitals is necessary to have high statistical power and sufficient heterogeneity among the subjects. However, combining data from multiple sources and performing statistical analysis on the union of the data poses privacy concerns [5].

A number of protocols have been proposed in the literature to address the privacy concerns; however, these protocols are either not completely private [6], [7] or are very demanding of the data holders involved in terms of complexity (extensive message passing among the participants and exponential computation complexity at each site) [8], [9].

We develop a privacy preserving linear regression using a semi-trusted third party (the Evaluator). We show that our approach has several desirable properties. The complexity at each site is independent of the number of involved sites, and the complexity for the Evaluator is linear in the number of sites. Private health information is preserved and the statistical outcome retains the same precision as that of raw data. Moreover, contrary to previous approaches, ours is complete. It not only calculates the linear regression parameters of a fixed model, but also includes model diagnostics and selection, which are the more important and challenging steps [5].

## 2. LINEAR REGRESSION

Linear regression consists of modeling the relationship between a set of variables referred to as attributes (or independent variables) and a response variable (or output variable). Fitting a linear regression model consists of a sequence of steps including estimation, diagnostics and model selection [10]. In what follows we give an overview of the steps involved, for more information the reader is referred to [10], [11]:

Assume that a dataset is composed of  $n$  input variables  $x_i \in \mathfrak{R}^{|D|}$  ( $\mathfrak{R}$  is the set of real numbers and  $D$  is the set of attributes), and  $n$  output variables  $y_i \in \mathfrak{R}$ . We denote by  $X$  the  $n \times D$  input matrix  $[x_i]$ , and by  $Y$  the  $n$  output vector.

Linear regression is the problem of finding the subset  $d \subseteq D$  of the attributes that affect and shape the response variable  $Y$ , and then learning the function  $f: \mathfrak{R}^{|d|} \rightarrow \mathfrak{R}$  that describe this dependency (of the output variables on the independent variables). Linear regression is based on the assumption that  $f$  is approximated by a linear map, i.e.  $y_i \approx f(x_i) = \beta x_i^{|d|} + \beta_0$

$i \in \{1, \dots, n\}$  for some  $\beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} \in \mathfrak{R}^{|d|}$ , where  $x_i^{|d|}$  is the

vector  $x_i$  restricted to the set of attributes  $d$  in question. In

linear regression literature, it is common to set  $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}$ , and

to augment every row  $x_i^{|d|}$  with 1 (so  $x_i^{|d|}$  is set to  $[1 \ x_i^{|d|}]$ ), with that the formulae above can be restated as:  $y_i \approx f(x_i) = \beta x_i^{|d|}$ . In what follows, we abuse the notation and use the superscript  $d$  instead of  $|d|$ .

Given a subset of attributes  $d \subseteq D$ , to learn the regression model (i.e. to learn the function  $f$ ) we need to find  $\beta$  such that  $y_i \approx \beta x_i^d$  best fit the dataset. The difference between the actual value  $y_i$  and the estimated  $\hat{y}_i = \beta x_i^d$  is referred to as the residuals:  $\varepsilon_i = y_i - \hat{y}_i$ . The goal in linear regression is to

find  $\beta$  that minimizes the square sum of the residuals ( $\sum_{i=1}^n \varepsilon_i^2$ ). The method used is referred to as the “least squares method” and it is equivalent to solving the following equation:

$$\beta = (X^{d^T} X^d)^{-1} X^{d^T} Y \quad (1)$$

The process of determining the best subset  $d \subseteq D$  is referred to as model selection. Model diagnostics are used to assess whether a fixed model (i.e. a regression model for a fixed  $d$ ) is proper. One statistics that reflects the goodness of fit for a fixed model is the adjusted  $R^2$  measure:

$$R_a^2 = 1 - \frac{(n-1) \sum_{i=1}^n (y_i - \hat{y}_i)^2}{(n-d-1) \sum_{i=1}^n (y_i - \bar{y})^2} \quad (2)$$

Where  $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$  is the output variable mean.

For a fixed model, once  $\beta$  is available,  $\hat{y}_i = \beta x_i^d$  can be calculated followed by  $R_a^2$ .

### 3. SETTING

In this paper, we consider the case of a dataset that is horizontally distributed among  $k \geq 2$  data warehouses (or data owners). The

different owners are interested in cooperatively studying the relationship between the independent and response variables, however they are not willing to share their data. This is known as privacy-preserving regression protocol [5]:

Let  $V$  be the matrix  $X$  augmented with column  $Y$ , i.e.  $V = [X : Y]$ . We consider the setting of  $k$  data holders,  $DW_1 \dots DW_k$ , each holding part of the matrix  $V$ . The division is assumed to be horizontal, i.e. each party holds a subset of the records of  $V$ . Denote by  $V_i$  the subset of matrix  $V$  held

by party  $i$ , then  $V = \begin{bmatrix} V_1 \\ \vdots \\ V_k \end{bmatrix} = \begin{bmatrix} X_1 : Y_1 \\ \vdots \\ X_k : Y_k \end{bmatrix}$ . In what follows,

we assume that  $X_1 = \begin{bmatrix} x_1 \\ \vdots \\ x_{n_1} \end{bmatrix}, \dots, X_i = \begin{bmatrix} x_{n_{i-1}+1} \\ \vdots \\ x_{n_i} \end{bmatrix}$ , and

$$X_k = \begin{bmatrix} x_{n_{k-1}+1} \\ \vdots \\ x_{n_k} \end{bmatrix}.$$

Before proceeding with an overview of the algorithm, we present two important properties for the horizontally distributed data:

1. For any  $d \subseteq D$ ,  $X^{d^T} X^d$  can be extracted from  $X^T X$  by removing all entries  $ij$  from matrix  $X^T X$  where either  $i$  or  $j$  do not represent a variable in  $d$ . The same applies for  $X^{d^T} Y$ .
2. Given that  $n > |D|$ , for any  $d \subseteq D$  we have that:

$$X^{d^T} X^d = \sum_{i=1}^k (X_i^{d^T} X_i^d) \text{ and that}$$

$$X^{d^T} Y = \sum_{i=1}^k X_i^{d^T} Y_i.$$

Hence Equation (1) is equivalent to:

$$\beta = \left( \sum_{i=1}^k (X_i^{d^T} X_i^d) \right)^{-1} \left( \sum_{i=1}^k X_i^{d^T} Y_i \right) \quad (3)$$

In what follows, we present a privacy-preserving linear regression protocol that uses a third party. The third party is referred to as the Evaluator. The Evaluator is assumed to follow the protocol correctly however if some data holders are corrupt, then the Evaluator will collaborate with them to obtain sensitive information about the data. Similarly, it is assumed that a corrupt data holder will correctly follow the protocol. We assume that up

to  $l-1$  data holders can be corrupt for some  $0 < l < k$ , thus, if  $l=1$  then all data holders are honest. The protocol is composed of 3 functions:

(a) Pre-computations, (b) a core regression protocol (referred to as  $CP$ ), and (c) an iterative protocol referred to as  $IP$ . The pre-computations are done once at the beginning of the protocol, then the  $IP$  protocol runs. It's role is to iterate over different values of  $d \subseteq D$ , calling  $CP$  for each such value of  $d$ . The  $CP$  protocol is performed by the evaluator with the collaboration of  $l$  out of the  $k$  data warehouses.  $CP$  takes as input a subset  $d \subseteq D$  and computes  $\beta$  and  $R_a^2$  for that given  $d$ . There are known iterative protocols for choosing the best subset  $d$  of independent variables [10], [11]. A common technique is to start with some basic set of attributes  $d_0 \subseteq D$  and find its corresponding  $\beta$  and  $R_a^2$ . Additional attributes can then enter the analysis one by one and the effect of each can be studied separately through  $R_a^2$ . An outline of the algorithm is presented in Figure 1, where the  $IP$  function and the algorithm flow are presented in details. The remaining functions will be presented in details in Section 6. For more information on regression the reader is referred to [5], [11].

#### 4. RELATED WORK

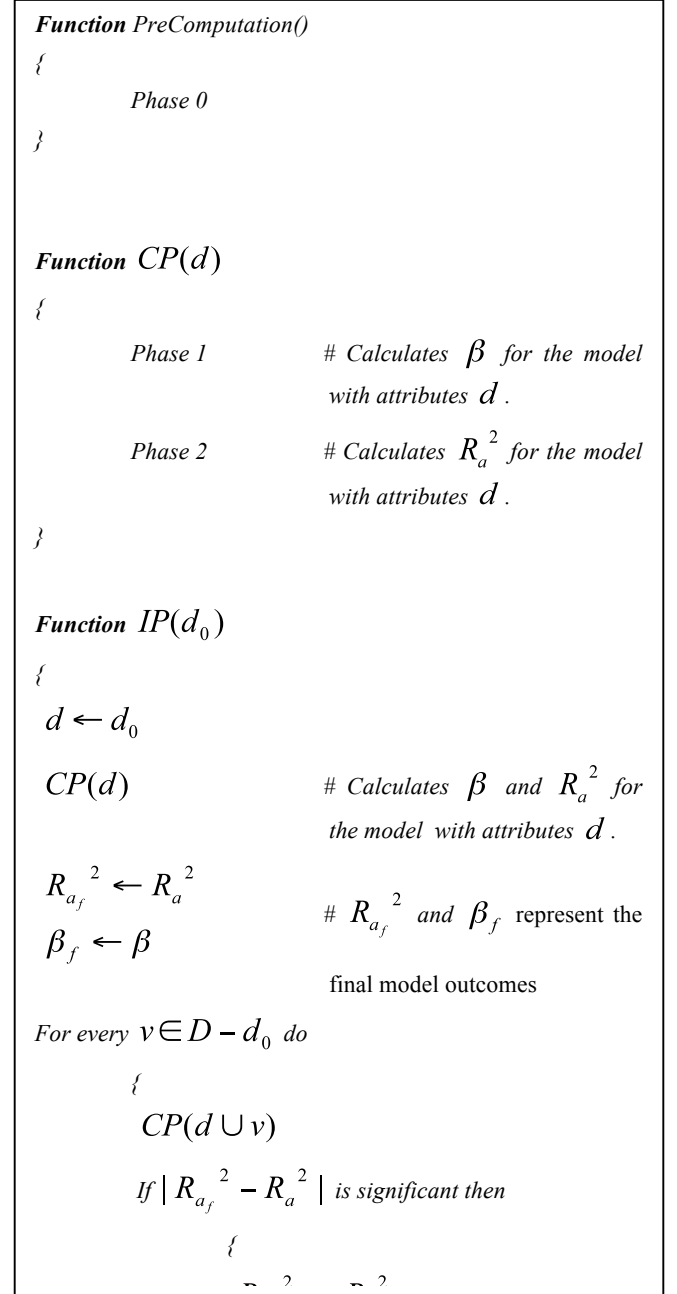
A number of protocols have been proposed for the collaborative computation of linear regression when data is horizontally distributed among the different parties. These protocols do not iterate over the attributes, they only offer fixed model solution (i.e. a solution for Equation (3)) and thus do not deal with model diagnostics.

The protocol in [7] suggests that the sites could share their local aggregate information. In other words site  $i$  would share with the other sites the aggregate values:  $X_i^T X_i$  and  $X_i^T Y_i$ . This way, each site can add the aggregate information to obtain  $X^T X$  and  $X^T Y$ , find the inverse of  $X^T X$ , then use Equation (1) to estimate the parameters of the regression. This method, although efficient, was criticized for being non-private as it shares local aggregate statistics [5], [8].

Another protocol due to Karr et al [6] suggests using secure multiparty computation in order to securely sum the local statistics:  $X_i^T X_i$  and  $X_i^T Y_i$ , the final sum  $X^T X$  and  $X^T Y$  is then shared among the different sites. This protocol, although efficient, was also deemed to be non-private [8].

Three more protocols have been suggested to solve this problem, two of these protocols [8], [9] use secret sharing and homomorphic encryption to privately calculate  $X^T X$ ,  $(X^T X)^{-1}$  and  $X^T Y$ , and then to multiply  $(X^T X)^{-1}$  and  $X^T Y$ . Both solutions make heavy usage of secure multiparty computation. As such, all data holders must remain online throughout the entire procedure. In both of these protocols, the

main computational component is the secure inversion of  $(X^T X)$  and their extensive use of the secure multiparty matrix multiplication protocol [12] extended to  $k$  parties. Each use of this  $k$ -party multiplication requires each pair of participants to



**Fig 1.** Algorithm flowchart

execute a 2-party secure matrix multiplication protocol. This amounts to a total of  $\frac{k(k-1)}{2}$  multiparty matrix multiplications. Such a  $k$ -party multiplication has each party executing a combination of  $k$  homomorphic matrix

multiplications and encryptions-decryptions under Paillier cryptosystem, as well as sending  $k$  matrices.

In [9], the inversion is done using an iterative method requiring two secure multiparty matrix multiplications for up to 128 iterations when using their settings for Paillier. This makes their protocol quite demanding on the data owners.

In [8], the authors present a generalization to  $k$  parties of the secure matrix sum inverse protocol of [12]. This allows them to compute the inverse in one step, which is an improvement on the inversion of [9], but their matrix inversion still requires around  $k^2$  secure 2-party matrix multiplications.

The third protocol was presented in [13], it uses additive encryption and Yao Garbled circuits. The protocol uses two non-communicating semi-trusted third parties. One party executes the algorithm, while the other holds encryption keys and generates garbled inputs. The additive encryption is used to privately compute  $X^T X$  and  $X^T Y$ , and Yao Garbled circuits to privately find the inverse of  $X^T X$ . While this solution does not require the involvement of all data holders, it requires two non-colluding semi-trusted parties each sharing part of the output. Moreover, the protocol requires the construction of garbled circuits. The construction of such circuits as well as its theoretical complexity were not tackled in the paper.

## 5. PRELIMINARY

In what follows we propose a privacy preserving regression protocol. We first introduce the properties of the public key cryptosystems that are used in our protocol.

Given a message  $m_i$ , we denote the ciphertext by  $c_i = Enc_{pk}(m_i)$  where  $pk$  is the public key used in the encryption. We will simply use  $Enc(m_i)$  when  $pk$  is clear from the context. In our protocol, we use Paillier cryptosystem [14] for the case where  $l = 1$  and threshold Paillier cryptosystem [15] when  $l > 1$  for their homomorphic properties.

Paillier cryptosystem is additively homomorphic, as such the sum of two messages can be obtained from their respective cyphertexts. For Paillier, this translates to  $Enc_{pk}(m_i + m_j) = Enc_{pk}(m_i) \times Enc_{pk}(m_j)$  [14]. Moreover, Paillier allows a limited form of homomorphic multiplication, in that we can multiply an encrypted message by a plaintext. It is done as follow:  $Enc(m_i)^{m_j} = Enc(m_i m_j)$ .

To simplify notation, given a matrix  $M$ , we let  $Enc(M)$  denote the entry-wise encryption of  $M$ . Thus, given two matrices  $A$  and  $B$ , the two properties of the Paillier encryption allows us to calculate the encrypted product  $Enc(AB)$  from  $Enc(A)$  and  $B$  as follows:

$$Enc(AB)_{ij} = \prod_k Enc(A)_{ik}^{B_{kj}}, \text{ where } M_{ij} \text{ represents the}$$

$ij^{th}$  entry in Matrix  $M$ . Similarly,  $Enc(AB)$  can be calculated from  $A$  and  $Enc(B)$ .

In a  $(n, t)$ -threshold cryptosystem, the secret decryption key is distributed among  $n$  different entities such that a subset of at least  $t$  of them are needed to perform the decryption [15], [16]. I.e. in order for the decryption to occur, at least  $t$  parties have to correctly perform their share of the decryption. The decryption shares are then combined to obtain the final decryption.

Note that our protocol will be using the threshold Paillier [15] cryptosystem when  $l > 1$ . This can be set up through a trusted party that will generate and distribute the public and secret keys. The trusted party can then erase all information pertaining to the key generation. If no such trusted party is available, the keys can be generated using secure multiparty computations [17]. Although this requires more computation overhead from each data owner, it only has to be done once. As such, it is an acceptable tradeoff.

In an  $(n, t)$ -threshold Paillier cryptosystem, the encryption is identical to the regular scheme. For the decryption, each party involved is required to compute the exponentiation of the cyphertext by their secret key. The product of these shares is then computed individually to proceed with the decryption. Since the validity of the decryption depends on the validity of the shares, threshold decryption protocols involve proofs of knowledge between each participant to prevent attacks by malicious parties. The complexity of the decryption is thus dominated by these proofs of knowledge [15].

We note that in our setting, each data owner will correctly execute the protocol even if they are corrupt, since they genuinely want the correct result. As such, we do not require the proofs of knowledge. This makes the threshold decryption only slightly more complex than the decryption in the setting  $l = 1$ .

## 6. PROTOCOL

In what follows, we present the Pre-computation function (also referred to as Phase 0), as well as the  $CP$  function which is composed of two phases, Phase 1, and Phase 2:

In Phase 0 some pre-computations are done. These computations are done once at the beginning of the Protocol.

The  $CP$  protocol is executed several times for different subsets  $d \subseteq D$ , it computes  $\beta$  and  $R^2_a$  for the given  $d$  with the collaboration of  $l$  out of  $k$  data warehouses, say  $DW_1, \dots, DW_l$ . Phase 1 of  $CP$  is dedicated to the calculation of the regression coefficients  $\beta$  and Phase 2 is dedicated for the calculation of  $R^2_a$ .

We assume that all the inputs are integer valued, due to the use of Paillier's cryptosystem. This is not a problem, as the data owners can multiply their data by a large non-private number. The effects of this multiplication can then be removed in intermediate/final results [12], [17].

Before presenting the protocol, we first start with some basic functions used throughout the protocol. The protocol will be presented for the general case where  $l > 1$ . When  $l = 1$ , some



steps can be optimized to slightly reduce the number of messages sent. These steps are presented after the protocol in a separate section. We assume that the total number of records  $n$  is public knowledge.

## 6.1 Basic Functions

The protocol uses several basic functions:

1. Creating Random Matrices, or  $CRM(d)$ :  $DW_1, \dots, DW_l$  and the Evaluator each generate a secret random  $d \times d$  matrix  $B_1, \dots, B_l, A$  respectively. We denote by  $B$  the product  $B_1 B_2 \dots B_l$ .
2. Creating Random Integers, or  $CRI$ :  $DW_1, \dots, DW_l$  each generate a secret random integer  $b_1, \dots, b_l$  respectively, while the Evaluator generates two random secret integers  $a$  and  $r$ . We denote by  $b$  the product  $b_1 b_2 \dots b_l$ .
3. Encryption and Decryption Functions for matrices, or  $Enc(M)$  and  $Dec(M)$ , respectively encrypts and decrypts the entries of a matrix  $M$ . This is an extension of the regular encryption and decryption functions on integers. Note that  $l$  data warehouses will be involved in the decryption.
4. Right Matrix Multiplication Sequence Function, or  $RMMS(B, Enc(M))$ , computes  $Enc(MB)$ . It is done as follow. The Evaluator sends  $Enc(M)$  to  $DW_1$ , who uses it to homomorphically compute  $Enc(MB_1)$  using its secret matrix  $B_1$ . The result is then sent to  $DW_2$ , who in turn computes  $Enc(MB_1 B_2)$ . The process repeats with  $DW_3, \dots, DW_l$ , and the result  $Enc(MB)$  is sent back to the evaluator.
5. Left Matrix Multiplication Sequence Function, or  $LMMS(B, Enc(M))$ , computes  $Enc(BM)$ . It is similar to  $RMMS(B, Enc(M))$ , but the order on the data warehouse is reversed.
6. Integer Multiplication Sequence Function, or  $IMS(b, Enc(m))$ , The Evaluator sends an Encrypted value  $Enc(m)$  to  $DW_1$ , who uses it to calculate the encrypted product  $Enc(mb_1) = Enc(m)^{b_1}$ , using its secret integer  $b_1$ . The result is then sent to  $DW_2$ , who in turn computes  $Enc(mb_1 b_2)$ . The process

repeats with  $DW_3, \dots, DW_l$ , and the result  $Enc(mb)$  is sent back to the evaluator.

## 6.2 Phase 0: Precomputations

At the beginning of this Phase, the Evaluator initiates  $CRI$ , thus the  $l$  data warehouses as well as the Evaluator generate a secret random integer each. This phase is composed of two main computations:

1. **Computations of  $Enc(X^T X)$  and  $Enc(X^T Y)$ :**

Each data holder  $DW_i$  locally computes her full local matrices  $X_i^T X_i$ , and  $X_i^T Y_i$ . She encrypts the matrices and sends them to the Evaluator. The Evaluator performs homomorphic additions and obtains

$$Enc(X^T X) = Enc\left(\sum_{i=1}^k X_i^T X_i\right), \quad \text{and}$$

$$Enc(X^T Y) = Enc\left(\sum_{i=1}^k X_i^T Y_i\right).$$

2. **Computation of  $\sum_{i=1}^n (y_i - \bar{y})^2$ :**

1. Each data warehouse  $DW_i$  sends their

$$\text{encrypted local aggregate } \frac{\phi_i}{n} = \sum_{i=n_{i-1}+1}^{n_i} \frac{y_i}{n}$$

to the Evaluator. The Evaluator homomorphically adds these values to get

$$Enc(\bar{y}) = Enc\left(\sum_{i=1}^n \frac{y_i}{n}\right), \quad \text{and then}$$

calculates  $Enc(a\bar{y}) = Enc(\bar{y})^a$ .

2. The Evaluator initiates  $IMS(b, Enc(a\bar{y}))$  and receives  $Enc(ba\bar{y})$ . He then initiates  $Dec(Enc(ba\bar{y}))$  to get  $ba\bar{y}$ .

3. The Evaluator computes  $Enc(b^2 a^2 \bar{y}^{-2})$  and initiates  $IMS\left(\frac{1}{b^2}, Enc(b^2 a^2 \bar{y}^{-2})\right)$

which results in  $Enc(a^2 \bar{y}^{-2})$ . The Evaluator then computes  $Enc(\bar{y}^{-2})$ .

4. The Evaluator propagates  $Enc(a\bar{y})$  to all data warehouses

- Each data warehouse computes

$$\begin{aligned} Enc(a\delta_i) &= Enc\left(\sum_{i=n_{i-1}+1}^{n_i} (-2ay_i\bar{y})\right) \\ &= \prod_{i=n_{i-1}+1}^{n_i} Enc(a\bar{y})^{-2} \sum y_i \end{aligned}$$

and  $Enc(\alpha_j) = Enc\left(\sum_{i=n_{i-1}+1}^{n_i} y_i^2\right)$ . Both are sent back to the Evaluator.

- The Evaluator computes

$$Enc(a\delta) = Enc\left(\sum_{i=1}^k a\delta_i\right) \text{ and recovers}$$

$Enc(\delta)$ . The Evaluator then proceeds to compute

$$\begin{aligned} Enc(\alpha) &= Enc\left(\left(\sum_{j=1}^k \alpha_j\right) - \delta + n\bar{y}^2\right) \\ &= Enc\left(\sum_{i=1}^n (y_i - \bar{y})^2\right) \end{aligned}$$

### 6.3 Protocol: $CP(d)$

First, given  $d$ , the evaluator extracts the encryptions of  $Z' = X^{d^T} Y$  and  $Z = X^{d^T} X^d$  from  $Enc(X^T X)$  and  $Enc(X^T Y)$  respectively. Then the Evaluator initiates  $CRM(d)$  and  $CRI$ .

### 6.4 Phase 1: Computing $\beta$

In this phase of the protocol, the Evaluator needs to compute  $Z^{-1}Z'$ . The steps are the following:

- The Evaluator computes  $Enc(ZA)$ , initiates  $RMMS(B, Enc(ZA))$  and receives  $Enc(ZAB)$ .
- The evaluator initiates  $Dec(Enc(ZAB))$  and receives  $ZAB$ .
- The evaluator computes  $B^{-1}A^{-1}Z^{-1} = (ZAB)^{-1}$  and calculates  $Enc(B^{-1}A^{-1}\beta)$  using  $B^{-1}A^{-1}Z^{-1}$  and  $Enc(Z')$ .
- The Evaluator obtains  $Enc(A^{-1}\beta)$  by initiating  $LMMS(B, Enc(B^{-1}A^{-1}\beta))$  and computes  $Enc(\beta)$  homomorphically.

- The Evaluator initiates  $Dec(Enc(\beta))$ , recovers  $\beta$  and sends it to all data warehouses.

### 6.5 Phase 2: Computing $Ra_2$

This is dedicated for the calculation of adjusted  $R^2$  measure given by equation (3):

- As each data warehouse  $DW_i$  knows  $\beta$ , they calculate their local residuals:  $\zeta_i = \sum_{i=n_{i-1}+1}^{n_i} (y_i - \hat{y}_i)$ , encrypt it and send it to the Evaluator. The Evaluator then adds the local residuals homomorphically to obtain  $Enc(\zeta) = Enc(\zeta_1) \times \dots \times Enc(\zeta_k)$ .
- The Evaluator computes  $Enc(r\zeta)$  and  $Enc(a\alpha)$ . He then initiates  $IMS(b, Enc(r\zeta))$  and  $IMS(b, Enc(a\alpha))$  and receives  $Enc(br\zeta)$  and  $Enc(ba\alpha)$  respectively.
- The Evaluator then initiates a decryption round for  $Enc(br\zeta)$  to obtain  $br\zeta$ , and uses it to compute  $ba\zeta = br\zeta \frac{a}{r}$ .
- The Evaluator now calculates  $Enc(R_a^2)$  homomorphically as follows:  $Enc(R_a^2) = Enc(1) - Enc(ba\alpha)^{\frac{(n-1)}{(n-d-1)ba\zeta}}$
- The Evaluator initiates  $Dec(R_a^2)$  and propagates the result to the different warehouses.

### 6.6 Special Considerations for the case $l=1$

For the case  $l=1$ , all the data owners are assumed to be incorruptible and all the decryption and obfuscation is delegated to one data warehouse, say  $DW_1$ . As such, the steps that initiate a multiplication sequence ( $RMMS$ ,  $LMMS$  or  $IMS$ ) followed by a decryption can be reversed and merged. In other words,  $DW_1$  can do the decryption first followed by the multiplication by its random number.

For example, in Phase 0, step 2.2 can be replaced by "The Evaluator sends  $Enc(a\bar{y})$  to  $DW_1$ , who decrypts to obtain  $a\bar{y}$ .  $DW_1$  then compute and send  $ba\bar{y}$  back to the evaluator." This will considerably reduce the complexity of  $DW_1$ 's computations when working with matrices.

Note that the role of  $DW_1$  can be assumed by another semi-trusted third party (STTP) if available. In such case, the Evaluator and the STTP will together compute the  $CP$  protocol. Both third

parties should follow the protocol correctly and should not communicate secretly outside the protocol.

## 6.7 Protocol Modification

In order to perform linear regression, the  $l$  participating data warehouses have to be online throughout the whole process. It would be ideal if the remaining  $k-l$  data warehouses could send their data at Phase 0, then stay offline throughout the remaining protocol. However this is not that case, these data warehouses have to participate in the calculation of  $R_a^2$  at each iteration of the  $CP$  protocol (refer to Step 2.1). With some changes to the protocol, it is possible for the data warehouses to send their full encrypted matrices  $Enc(X_i)$  and  $Enc(Y_i)$  to the Evaluator at the start of the protocol (i.e. in Phase 0) then stay offline for the whole process afterwards. The Evaluator can use these encrypted matrices to perform Step 1 of Phase 2 without the involvement of the  $k-l$  data warehouses. In other words, the Evaluator can calculate  $Enc(\xi_i)$  using  $\beta, Enc(X_i)$ , and  $Enc(Y_i)$  as follows:

$$Enc(\hat{y}_i) = \prod_{j=1}^d Enc(x_{ij}^d)^{\beta_j} = Enc(\beta x_i^d)$$

and

$$\begin{aligned} Enc(\xi_i) &= Enc\left(\prod_{i=n_{i-1}+1}^{n_i} Enc(y_i) \times Enc(-\hat{y}_i)\right) \\ &= Enc\left(\sum_{i=n_{i-1}+1}^{n_i} (y_i - \hat{y}_i)\right) \end{aligned}$$

The problem with this modification is that the data warehouses would give out their local number of records,  $n_1, \dots, n_k$ . If this is considered private information, then the original protocol has to be followed. Note that this modification requires considerably more space and complexity at the Evaluator side as the encrypted  $n$  records have to be stored and then used to calculate  $Enc(\xi_i)$ .

## 7. PRIVACY DISCUSSION

In this section we study the privacy of our protocol and show that no party can learn any information other than the final results of the regression. We assume that  $l-1$  parties are corruptible and that the total number of records,  $n$ , is known.

Since Paillier is semantically secure [14] and since we need  $l$  parties to complete decryption, no information can be gained from the ciphertexts exchanged throughout the protocol. We note that when  $l=1$ , all but one of the ciphertexts sent to  $DW_1$  for obfuscation will be decrypted at the next step, so  $DW_1$  can only gain additional knowledge from a ciphertext if he decrypts  $Enc(a\alpha)$  in Phase 2.2. As such, we will look at the decrypted values each party obtains.

In Phase 0, all the values are encrypted except for  $ba\bar{y}$ , that the  $l$  active data owners and the evaluator receive. If the corrupted data owners and the evaluator collaborate, they can remove  $a$  and at most  $l-1$   $b_i$ 's. They can obtain  $b^l \bar{y}$ , but since  $b^l$  is a random unknown integer, no information about  $\bar{y}$  can be recovered. In the case where  $l=1$ , the active data owner obtains  $a\bar{y}$  from the decryption, but since  $a$  is random, no information is gained. The same reasoning is true for Phase 2, where all the information is encrypted except for  $br\bar{\xi}$ , which is always obfuscated by at least one random integer. In the case where  $l=1$ ,  $DW_1$  can also obtain  $a\alpha$  by performing an extra decryption, but no information about  $\alpha$  or  $\bar{\xi}$  can be gathered since both are obfuscated by different random integers.

In Phase 1, all the matrices are encrypted except for  $ZAB$  that all active party and the evaluator obtain in step 1.2. The evaluator and the corrupted party can obtain  $ZAB'$ , where  $B' = B_1 B_2 \dots B_i$  and  $DW_i$  is an incorruptible party. Since  $B'$  is random, the evaluator cannot recover  $B_i$  or  $Z$ , even while knowing  $AB_1 B_2 \dots B_{i-1}$ .

We thus have that all the values in the protocol are either encrypted, obfuscated by some random element or sent to all the parties. As such, our scheme is secure and private, since no party can learn additional information from the protocol other than the final result of the computations.

## 8. COMPLEXITY

In this section, we evaluate the computational complexity and the amount of messages sent during one iteration of our protocol. We will evaluate the individual burden of each participating party as well as the total complexity. Let  $d$  be the number of attributes used for a given iteration and let  $D$  be the total number of attributes considered.

Our result will be given using some basic functions as the units. More specifically, we will give how many encryptions, decryptions, homomorphic multiplications (HM) and homomorphic additions (HA).

If we are using an instance of Paillier with modulus  $m^2$ , we have that HA is equivalent to multiplying two integers modulo  $m^2$ , while HM is equivalent to computing an exponentiation modulo  $m^2$ . It follows that an encryption is equivalent to 2HM and 1HA, while a standard decryption is essentially 1HM [14].

Finally, in our setting, a  $(k, l)$ -threshold decryption is equivalent to having each of the  $l$  involved party compute one HM and  $l$  HA as well as send  $l-1$  messages. It follows that, since  $l \ll m$ , HM dominates the decryption and the  $l$  HA have a negligible impact on the complexity [18]. As such, we can reasonably assume that  $(k, l)$ -threshold decryption is bounded

above by a constant computational complexity of 2HM, making it only slightly more expensive than standard decryption.

We will start by evaluating the complexity of the basic functions used throughout the protocol.

1.  $Enc(M)$  and  $Dec(M)$ : These functions involve  $d^2$  encryption and decryptions, respectively. If the result is sent, it also requires a total of  $d^2$  and  $(l^2 + 1)d^2$  messages, respectively.
2.  $RMMS(B, Enc(M))$  and  $LMMS(B, Enc(M))$ : In these functions, each party sends  $d^2$  messages to exactly one other party, for a total of  $(l + 1)d^2$  messages. The data owner  $DW_i$  has to homomorphically compute  $Enc(MB' B_i)$ , with each entry of this matrix requires at most  $d$  HM and  $(d - 1)$  HA. Thus, the whole matrix requires at most  $d^3$  HM and  $d^2(d - 1)$  HA for each  $DW_i$ .
3.  $IMS(b, Enc(m))$ : Each party sends one message to exactly one other party, for a total of  $l + 1$  messages. Each data owner  $DW_i$  has to homomorphically compute  $Enc(mb' b_i)$ , which requires one HM.

We now evaluate the complexity of each phase, starting with Phase 0. Recall that this phase is all about pre-computations. As such, it will not affect the complexity of an iteration of Phase 1 and 2.

1. Each data owner first computes  $Z_i$  and  $Z'_i$ . They are then required to send  $D^2 + D + 2$  encryptions (steps 0.1, 0.2.1 and 0.2.5) and compute 1 HM in phase 0.2.5. Each of the  $l$  active data owners also participate in 2IMS and 1 decryption.
2. The evaluator has to perform a total of  $(k - 1)(D^2 + D + 3) + 2$  HA, 3HM and 1 encryption. He sends a total of  $2k + 3$  messages.

We now evaluate the complexity of the main protocol, starting with Phase 1. We will not take the generation of the random integers and random matrices into account, since they can be generated and stored ahead of time. In Phase 1, the passive data owners do not participate. Each of the  $l$  active data owners participate in 2MMS and  $2d^2$  decryptions.

The evaluator performs  $d^2$  encryptions, inverts one plaintext matrix, and sends messages in 2MMS and  $2d^2$  decryptions. He also computes a total of  $(d^3 + 2d^2)$  HM and  $(d^2 + 2d)(d - 1)$  HA in steps 1, 3 and 4.

Phase 1 involves a total of  $2d^2$  decryptions,  $(3d^3 + 2d^2)$  HM,  $(3d^2 + 2d)(d - 1)$  HA and one matrix inversion. A total of  $(l^2 + 2l + 3)d^2 + (l^2 + 1 + k)d$  messages are sent among all the parties.

In Phase 2, all data owners compute  $X\beta$  and  $\xi_i$ , and perform one encryption. Each of the  $l$  active data owners also participate in 2IMS and 2 decryptions. The Evaluator computes a total of  $k$  HA and 3HM. The Evaluator also performs a plaintext multiplication in step 2.3. A total of  $2(k + l + l^2 + 1)$  messages are sent among all the parties.

As such, assuming a decryption is at most 2HM, the final complexity of  $CP(d)$  for each participating party is bounded above by the following.

1. All data owners: 2 matrix multiplications, 1 encryption. Sends 1 message.
2. Active data owners additionally have:  $2(d^3 + d^2 + 1) + (2d^2 + 2)$  HM,  $2d^2(d - 1)$  HA. Sends  $(l + 1)(d^2 + d + 1) + 1$  messages.
3. The Evaluator: 1 matrix inverse, 1 plaintext multiplication,  $(d^3 + 2d^2 + 3)$  HM,  $(3d^2 + 2d)(d - 1) + k$  HA. Sends  $(2d^2 + 2d + kd + 4)$  messages.

As can be seen from this evaluation, if we fix the dimension  $d$ , the total complexity of the scheme is linear in  $k$ , while the total number of messages is  $O(l^2 + k)$ . The Evaluator absorbs most of the computational complexity, leaving the data warehouses with a complexity depending only on the size  $d$  of the matrices, if we assume  $l \ll m$ . This shows that our protocol allows the data owners to greatly reduce the computational power needed for a multiparty regression by making use of a STTP (the Evaluator).

Finally, we will compare the complexity of our scheme to that of the schemes of [9] and [8] for each individual participants. For this we shall look mostly at the secure multiparty matrix multiplication protocol of [12]. In the 2-party case, one party has to compute about  $3d^2$  HM and  $d^2$  HA for encryption and decryption while the second party has to execute about  $d^3$  HM

and  $d^3$  HA for the homomorphic matrix multiplication and share splitting. As such, in the  $k$ -party protocol we can expect an average of  $\frac{k}{2}(d^3 + 3d^2)$  HM,  $\frac{k}{2}(d^3 + d^2)$  HA and  $kd^2$  messages for each participating member.

This multiparty secure matrix protocol is executed at least 2 times in [8] and up to 248 times in [9] when computing the inverse of  $Z$ . We note that, for any  $k$ , our complete protocol  $CP(d)$  involves less computational burden and messages for each party than a single matrix inversion in [8] or [9]. This is due to the fact that the individual complexity in our protocol is independent of  $k$  for all but the Evaluator.

## 9. CONCLUSIONS AND FUTURE WORK

We presented a practical system that performs linear regression for a large number of data warehouses without learning anything about the data apart from the regression parameters and diagnostics.

Different from existing approaches, our approach is complete. It not only calculates the parameters  $\beta$  of a fixed model, but also includes model diagnostics and selection, which are more important and more challenging steps[5].

Our model is superior in terms of complexity on the data holders end as the Evaluator absorbs most of the regression complexity.

We are currently in the process of applying the protocol on the union of three datasets from the state of Pennsylvania (over 1.5 million records). The study aims to find the attributes that affect surgery completion times and come up with recommendations. The trusted third party we will be using is the IBM Cloud at Western University.

## 10. REFERENCES

- [1] E. M. Stahl, *Emergency Department Overcrowding: Its Evolution and Effect on Patient Populations in Massachusetts*. ProQuest, 2008.
- [2] D. S. Kc and C. Terwiesch, "Impact of workload on service time and patient safety: An econometric analysis of hospital operations," *Manag. Sci.*, vol. 55, no. 9, pp. 1486–1498, 2009.
- [3] G. P. Pisano, R. M. Bohmer, and A. C. Edmondson, "Organizational differences in rates of learning: Evidence from the adoption of minimally invasive cardiac surgery," *Manag. Sci.*, vol. 47, no. 6, pp. 752–768, 2001.
- [4] R. Reagans, L. Argote, and D. Brooks, "Individual experience and experience working together: Predicting learning rates from knowing who knows what and knowing how to work together," *Manag. Sci.*, vol. 51, no. 6, pp. 869–881, 2005.
- [5] J. Vaidya, C. W. Clifton, and Y. M. Zhu, *Privacy Preserving Data Mining*. Springer, 2005.
- [6] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Secure regression on distributed databases," *J. Comput. Graph. Stat.*, vol. 14, no. 2, 2005.
- [7] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proceedings of the 4th SIAM International Conference on Data Mining, 2004*, vol. 233.
- [8] K. El Emam, S. Samet, L. Arbuckle, R. Tamblyn, C. Earle, and M. Kantarcioglu, "A secure distributed logistic regression protocol for the detection of rare adverse drug events," *J. Am. Med. Informatics Assoc. JAMIA*, vol. 20, no. 3, pp. 453–461, May 2013.
- [9] R. Hall, S. E. Fienberg, and Y. Nardi, "Secure multiple linear regression based on homomorphic encryption," *J. Off. Stat.*, vol. 27, no. 4, p. 669, 2011.
- [10] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*, vol. 821. Wiley, 2012.
- [11] G. A. Seber and A. J. Lee, *Linear regression analysis*, vol. 936. John Wiley & Sons, 2012.
- [12] S. Han and W. K. Ng, "Privacy-preserving linear fisher discriminant analysis," in *Advances in Knowledge Discovery and Data Mining, Springer, 2008*, pp. 136–147.
- [13] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-Preserving Ridge Regression on Hundreds of Millions of Records," 2012.
- [14] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptology—EUROCRYPT'99, 1999*, pp. 223–238.
- [15] C. Hazay, G. L. Mikkelsen, T. Rabin, and T. Toft, "Efficient rsa key generation and threshold paillier in the two-party setting," in *Topics in Cryptology—CT-RSA 2012, Springer, 2012*, pp. 313–331.
- [16] Y. Desmedt, "Threshold cryptosystems," in *Advances in Cryptology—AUSCRYPT'92, 1993*, pp. 1–14.
- [17] T. Nishide and K. Sakurai, "Distributed paillier cryptosystem without trusted dealer," in *Information Security Applications, Springer, 2011*, pp. 44–60.
- [18] H. Cohen, *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.

# Data Anonymization: The Challenge from Theory to Practice

Ting Yu

Department of Computer Science  
North Carolina State University  
Raleigh, North Carolina, USA  
yu@csc.ncsu.edu

## BIO

Ting Yu is an associate professor in the Department of Computer Science of North Carolina State University, and a senior scientist in the cyber security group of Qatar Computing Research Institute (QCRI). His main research areas are in data privacy and anonymization, trustworthy information in open systems and trust management. He obtained his Ph.D. in computer science from the University of Illinois at Urbana Champaign in 2003. Ting Yu is a recipient of the NSF CAREER Award in 2007 for trust and privacy management in social networks, and a recipient of the scholarship of K.C. Wong Education Foundation, Hong Kong in 2010.

(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0

*7th International Workshop on Privacy and Anonymity in the Information Society (PAIS'14)* March 28, 2014, Athens, Greece

# A Privacy Preserving Model for Ownership Indexing in Distributed Storage Systems\*

Tiejian Luo  
University of Chinese  
Academy of Sciences  
tjluo@ucas.ac.cn

Zhu Wang  
University of Chinese  
Academy of Sciences  
wangzhubj@gmail.com

Xiang Wang  
University of Chinese  
Academy of Sciences  
wangxiang11@mails.ucas.ac.cn

## ABSTRACT

The indexing technique in distributed object storage system is the crucial part of a large scale application, where the index data structure may be published in many nodes. Here arises a problem on preserving the privacy of the ownership information while supporting queries on item locations with limited index space. Probabilistic data structure, such as the bloom filter which records the location of each item in distributed nodes, is one of the promising solutions. The data structure uses a hashed vector to index items on the nodes. In this paper we propose a Lightweight Bloom filter Array (LBA) indexing model which is compact in size and preserves ownership privacy. To tackle with the problem of examining wrong nodes in the lookup process, we find an optimal storage ratio of the bloom filters and reduce its false positive rate based on the observation of the user's access behavior in Internet applications. We use experiments to verify our proposed solution. In our experiment, the dataset consists of one billion items distributed in one hundred data nodes. The experiments show that our model can reduce the false checking times and save the index space significantly.

## 1. INTRODUCTION

With the rapid growth of the Internet, many online applications have been based on distributed storage systems which are composed of many single storage nodes. When a request for a certain item arrives at the system, the first step is to find which node contains the item. The indexing service, which is capable of recording ownership relation between nodes and items, is a key component in distributed systems. The node that holds the service is called the index node. It can be either a storage node that have the index function the same time as the storage role or an exclusive node that

\*(c) 2014, Copyright is with the authors. Published in the Workshop Proceedings of the EDBT/ICDT 2014 Joint Conference (March 28, 2014, Athens, Greece) on CEUR-WS.org (ISSN 1613-0073). Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

is only responsible for the indexing service. Because of the large total number of items, each storage node in the system is usually in charge of many items. However, in many distributed systems, the indexing data structure has to be deployed in many (even all) nodes in order to support fast lookup and high scalability. Therefore the index has to be very compact in size in order to be stored in the index nodes' memory. Moreover, in many distributed systems, an item can be a video slice, file fragment or even a piece of record. It needs several times of item lookup in the table to finish a meaningful service. Therefore, we need a high performance indexing technique with limited space consumption in distributed systems.

Private information grows with the increase of data volume and service type. Index data structure has to support the queries for resident nodes of an item while keeping the entire ownership information away from unknown requests. Since distributed systems may have more than one index nodes, it is sometimes unavoidable for the information transportation and indexing publishing, and hence unpredicted interception. In those cases, the security of the information stored in the index may be threatened. Compact, fast and secure index will become a key component in distributed applications.

Bloom filter[2] is a space-efficient probabilistic data structure for item representation and lookup in a set. When indexing space is limited, i.e. in the memory, the data structure offers fast item lookup with a low false positive rate. Many distributed systems that emphasize time efficiency are using bloom filters as their indexing technique when a small false positive rate is tolerable[8]. The data structure uses hash functions to map items onto several positions on a bit vector. It only stores the hashed bits and allows for hash collisions. The actual data is not stored on the vector. Without the knowledge of hash functions, the ownership information cannot be obtained from the vector. In this way, the privacy is preserved. Actually, many online systems[1, 4, 3] are using bloom filters for secure index. In our paper we try to optimize their usage with reference to user behavior.

The observation of user behavior indicates that in many applications, a small number of items attract a major part of user access. The phenomenon inspires us to be selective in index construction when space is limited. In this paper we provide a lightweight mechanism for bloom filter usage. The lookup procedures are given to guarantee effective item



locating.

The rest of the paper is organized as follows. Section 2 describes the related work of our research. In Section 3, we give the lightweight bloom filter construction and item lookup procedures. Experimental results are shown in Section 4. Finally, we present the conclusion and future work in Section 5.

## 2. BLOOM FILTER AND PURE BLOOM FILTER ARRAY (PBA) INDEX

### 2.1 Bloom Filter

Bloom filter[2] works as an index which records all elements of a set. We may assume that the set  $S = \{x_1, x_2, \dots, x_n\}$ , which consists of  $n$  elements. A Bloom Filter vector (BFV), which consists of  $m$  bits, is used to represent elements of set  $S$ . All bits of the vector are set to zero initially. For each element, the algorithm uses  $k$  hash functions  $\{h_i\}_{i=1 \dots k}$  to map the element onto  $k$  positions of the vector and sets the bit on the position to 1. The  $k$  functions, ranging from 1 to  $m$ , are independent from each other and can map elements of the set  $S$  to a random place on the vector. During the insertion period, the algorithm maps all elements of the set to load the BFV with all the information of the elements.

In lookup procedure which we want to check whether an element  $y$  belongs to the set  $S$ , the algorithm uses the same hash functions to map  $y$  onto  $k$  locations and check whether all  $h_i(y)$  equal to 1. If the answer is no, we conclude that  $y$  doesn't belong to  $S$ , otherwise, we say  $y$  belongs to  $S$ . The time complexity of bloom filter lookup is constant.

It needs to be mentioned that there is a probability that elements don't belong to  $S$  be judged as inside  $S$  by BF. That is to say, BF has a false positive rate. Research[6] shows that the false positive rate can be represented as follows:

$$f_{FP} = (1 - e^{-\frac{kn}{m}})^k \quad (1)$$

Study[6] also shows that  $f_{FP}$  reaches minimal value when

$$k = \frac{m}{n} \ln 2 \quad (2)$$

Then the false positive is minimized

$$f_{FP} = 0.6185^{\frac{m}{n}} \quad (3)$$

Since the bloom filter does not store the actual data, ownership information cannot be revealed without the knowledge of hash functions. The algorithm can be used as secure index in online applications[1, 4, 3]. Due to its simple structure and smooth integration characteristic, the mathematical format allows considerable potential improvement for system designers to develop new variations for their identical application requirements. In this paper, we focus on fast object lookup in distributed systems.

## 2.2 Pure Bloom Filter Array for Distributed Data Storage Index

Many distributed systems use Pure Bloom filter Array (PBA)[10] to support item index and lookup. The approach consists of a two-stage process: indexing building and item locating.

*Index Building.* For each node of the system, the index node builds a bloom filter for representing all of its items. These Bloom filters are loaded with all the items in the entire system and can act as an indexing system.

*Item Locating.* The object locating process is described below: when a query for a certain item arrives on the index node, the node first uses the bloom filters to find the approximate membership relations: it calculates with the bloom filter of each node and collects the results. The negative result of a certain bloom filter means that the queried item doesn't exist on the related node. The positive result means that the queried item exists on the node with a probability of  $1 - f_{FP}$ . Then the system queries the actual node whose bloom filter check result is positive to check whether the queried item exists in the node. In that way, the false positive occurrence is finally eliminated. Since the bloom filters have a constant time complexity, the method can reduce lookup time remarkably.

## 3. LIGHTWEIGHT BLOOM FILTER ARRAY FOR DISTRIBUTED STORAGE INDEX

### 3.1 Lightweight Bloom Filter Design

User behavior observation indicates that access for items varies between different objects. In many applications, the access frequency can be observed accurately. If ordered by access frequency, the top ranked items attract a large portion of user visits while the low ranked items absorb a very little part. That phenomenon shows us a way to increase bloom filter space use efficiency. Each node in the system builds a bloom filter for indexing the items of the node. It selects the highly ranked items and inserts them into the bloom filter. The bloom filters on the nodes forms an array, which is capable of recording item ownership information on nodes and plays the role of a distributed index. Though the total index space is limited, the data structure has a lower load and therefore a lower false positive rate. Queries for highly ranked (popular) items will have a more accurate response. Queries for low-ranked items will not receive a positive response from the bloom filter index. In those cases the system uses traditional lookup method to find the queried item directly in the storage nodes. Since most queries are for popular items, the overall false positive rate of the index can be reduced. The detailed bloom filter improvement method is described below. In that way, the index can perform with lower space consumption and preserve ownership privacy.

*Index Building.* The system sets a load factor  $\beta$ , which is the ratio of the loaded item to the total item number. Each node in the system orders the items by their access time. Then it inserts the items one by one from rank 1 until it reaches the load threshold. The bloom filters are gathered to form an array, which represents the ownership relation between items and nodes, and stored on the index nodes. For a system of totally  $N$  items, the bloom filter arrays index  $\beta N$  items.

*Item Locating.* When a query for a certain item arrives, the index node first calculates with the bloom filter of each node and collects the results. One possible situation after the calculation is that there is at least one positive result, it means that the queried item exists on the node with a probability of  $1 - f_{FP}$ . Then the system first queries the actual node whose bloom filter check result is positive to verify whether the queried item exists in the node. If it does exist on one of the positive nodes, the lookup procedure stops; otherwise it continues to check on the remaining negative nodes until it finds the queried item. The other possible situation after bloom filter calculation is that there is no positive match. Under that circumstance, the system looks up the item in each unsearched node directly. The lookup procedure is shown in Fig.1.

In the two steps of the lookup procedure, the bloom filter calculation takes place in index nodes' memory. The time consumption is rather low considering the  $O(C)$  time complexity of bloom filters. The vast majority of time cost comes from the node lookup process, in which the index node communicates with the storage node and the storage node check in its disk for queried items. So the time consumption is approximately linear to the average checking times of a query. In the many check in nodes, only one of the checking procedure can find the needed item. The rest nodes checking end without a match and waste a lot of time and system resource. Those redundant (false) checking times are the key factor that lowers system performance. The occurrence that the system looks up a query in a wrong node and finds no result is called the *false checking* in nodes.

### 3.2 Selection of Popular Items

In the design of LBA, the algorithm stores the top  $\beta \times 100\%$  popular items in the bloom filter index. In actual processing, each node keeps a list of all its popular items and builds the lightweight bloom filter of its own. Then it transmits the index to the index nodes, where all bloom filters are stored in.

The selection of the popular items follows the same procedure of that with cache. The goal is to find the top  $\beta \times 100\%$  items in each node efficiently with high accuracy. The cache selection and update algorithms are adequate for that task. Actually in [10], the authors use LRU cache scheme for selecting popular items.

It needs to mention that the item selection process takes place in each storage node, just like the cache does. It does not occupy the resource of the index node, which is responsible of storing the bloom filters already built on the storage nodes. Also, the index node does not have the entire knowledge of the ranking of items on the storage nodes.

### 3.3 Dynamic Items and Renewal Process

In online systems, the access pattern of items and visit frequency varies over time. A "hot" item may become unpopular after a certain period of time. The updating of popular list on each storage node follows the same way as caches do. After reaching a certain threshold, the node rebuilds the bloom filter index in the same way as [10] does. Then it transmits the new filter to the indexing node. In update

mechanism, we use the mechanisms given in [10] and [9]. The effect of that method is given in [9].

## 4. EXPERIMENTAL EVALUATIONS

In this section we use experiments to show the index performance. The items and nodes are synthetic data. In all experiments we set node number  $s=100$ , the total number of items  $N = 10^9$ . The items are scattered randomly among  $s$  nodes, so each node has approximately  $n = 10^7$  items. The probability that an object be allocated in one node is identical among all servers. The total query number  $Q = 10^5$ .

### 4.1 Queries

Observations show that several "hot" items attract a majority of user access, as stated in Zipf's law[7, 5]. In the experiment we assume that access for the entire corpus follows Zipf's distribution with total number  $N = 10^9$ . The queries reflect the real popularity of the corpus, so the queries follow the same distribution as the corpus. Actually, queries are generated automatically as a sampling set of Zipf's law with parameter  $N = 10^9$ .

The real query count of top 10000 hot items are plotted in Fig.2.

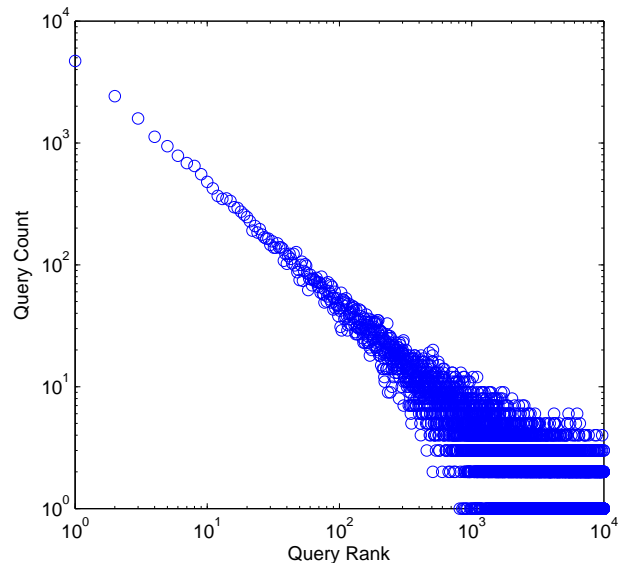


Figure 2: Query count for top 10000.

The figure is in a log-log scale. We can see that the first half part of the figure is a nearly straight line with slope=-1. That indicates the queries follow the Zipf's law. The second half consists of some irregular points because of the low value of the actual count.

### 4.2 Node Construction

In actually processing of bloom filter construction in storage node, we first order all items in the node by its popularity rank and pick the first  $\beta n$  items. Then we insert those items into its bloom filter. For easy deployment in the memory of the node, the bloom filter needs to be compact in size. In the experiment we set the length of the bloom filter vector

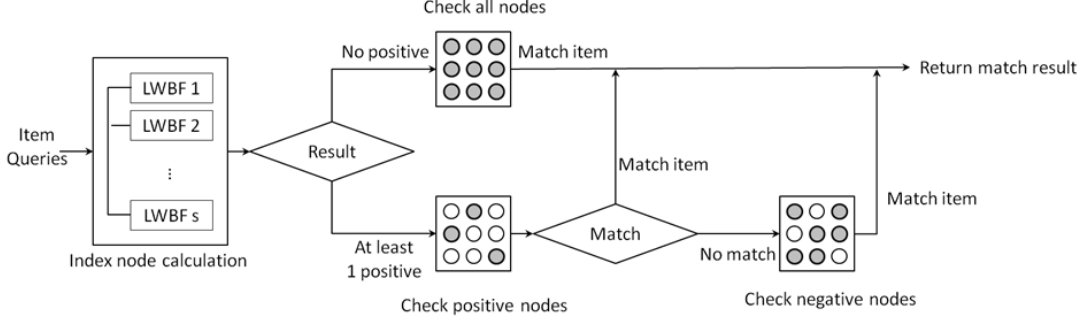


Figure 1: Item lookup procedure.

$m=33554432$  (use 3.36 bits for one item). The hash number of the bloom filters reaches the nearest integer of its optimal value in (2).

### 4.3 Experiments with Different $\beta$

In the experiment we want to analyze the impact of  $\beta$ . First, we want to know whether our proposal can actually improve the bloom filter index performance. Second, we are going to find out what the optimal load factor is in the lightweight bloom filter.

The range of  $\beta$  is  $\{0.1, 0.2, \dots, 1\}$ . Here  $\beta=1$  means that the bloom filters are fully loaded, which is equivalent to pure bloom filter array. For each  $\beta$ , we build the bloom filter index in each node and lookup all  $Q$  queries until we find a match. In the experiment, we count the total false checking times when we finish serving all the queries.

In order to find if our proposed method have a positive effect, we use the PBA approach and the direct lookup approach (in which nodes are checked one by one to find a queried item without using index) for a comparison. In the following experiment we repeat the indexing and querying procedure separately with the same system environment: index objects, queries, nodes, etc. Then we count the false checking times of each method. The false checking times in each experiment is plotted in Fig.3.

In the figure we can see that both LBA and PBA have an impressive improvement over direct lookup method (DL) considering false checking rate. That will reduce the overall system workload. Comparing PBA and LBA, we find that the LBA performs better than the PBA and reaches its optimization when  $\beta=0.4$ . The two methods come near when  $\beta$  approaches 1. When  $\beta=1$ , the LBA is fully loaded and is equivalent to PBA. The false checking rate of LBA reaches only 30 percent of that in PBA at optimal  $\beta$ . That will have a direct impact on system performance. Under that system parameter, when top forty percent popular objects are loaded, the false checking rate is 5.87, which means that on average the system will check 5.87% of all nodes before finding a query.

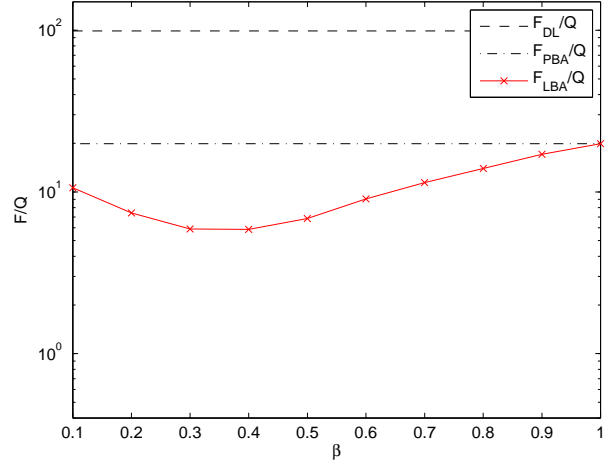


Figure 3: Overall false checking in node.

### 4.4 Space Consumption

Compared with the PBA, LBA can be more accurate when the index memory cost is very low. Equivalently, when reaching the same accuracy, the LBA can use less space than PBA. Take the example in the experiment, with one billion items distributed in one hundred nodes, a query will be checked on 5.87% of all nodes before finding its right location. The space needed for the indexing structure is only about 400MB using LBA (3.35 bits for one item), which is affordable in many distributed nodes. If we use PBA method instead, in order to achieve the same accuracy, we need about 700MB memory space. The space needed has reduced by 43%.

### 4.5 Distribution of False Checking Times

In the experiment, we record the false checking times of each query. Fig.4 shows the experimental false checking distribution when  $\beta=0.4$ . We see that most queries will find their match after ten false checking times. Some queries will have to go through all nodes to find a match.

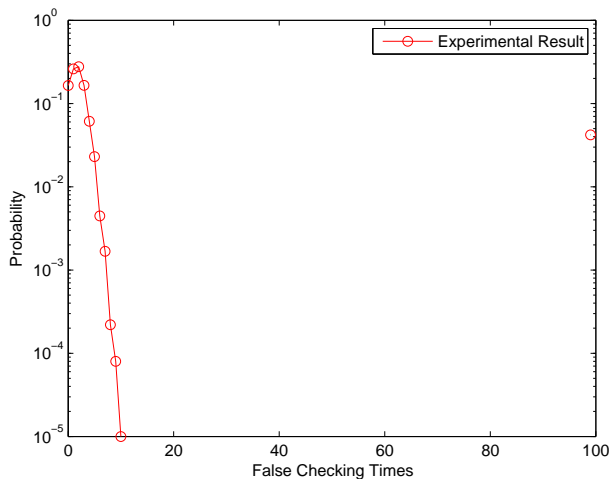


Figure 4: False checking ( $\beta=0.4$ ).

Fig.5 shows the experimental false checking times distribution with different  $\beta$ . We can see that for every  $\beta$ , the false positive rate concentrates on two areas: the low false checking area (Bernoulli distribution) and s-1. When  $\beta$  increases, the mean of Bernoulli distribution increases while the false checking times s-1 decreases (when  $\beta=1$  the false checking on s-1 is 0).

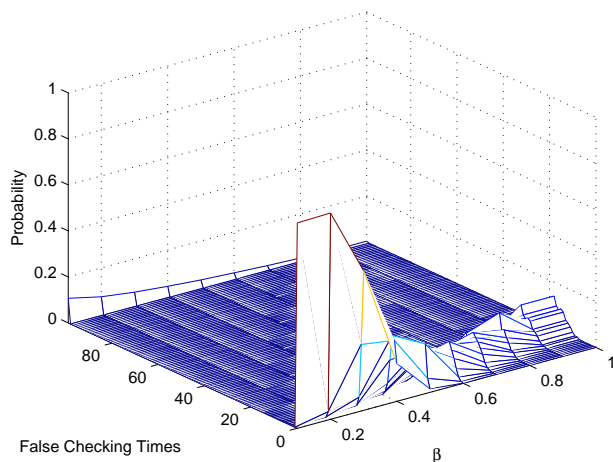


Figure 5: False checking distribution.

## 5. CONCLUSION

In the paper we have presented a privacy preserving model for ownership indexing in distributed storage systems - the lightweight bloom filter array based on the observation that user access for items varies from case to case in many Internet applications. We have pointed out a problem in the

secure index - support fast ownership query with limited index space. We have changed the traditional bloom filter construction method by ranking the items and putting only a part of top visited items into the bloom filter array. Then the experimental evaluation has been conducted to show that the mechanism can reduce false checking times by 70 percent. We have demonstrated that our algorithm can improve the system performance while preserving the privacy of ownership relation in distributed systems when the index nodes' memory is limited.

The future work includes adopting the new algorithm to more complex indexing systems and adding new functions to the algorithm, i.e. item deletion. The improvement of lookup procedure will also continue in the development of the algorithm.

## 6. REFERENCES

- [1] M. Bawa, R. J. Bayardo Jr, R. Agrawal, and J. Vaidya. Privacy-preserving indexing of documents on the network. *The International Journal on Very Large Data Bases*, 18(4):837–856, 2009.
- [2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426, 1970.
- [3] E.-J. Goh. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.
- [4] F. Jian-ming, X. Ying, X. Hui-jun, and W. Wei. Strategy optimization for p2p security using bloomfilter. In *International Conference on Multimedia Information Networking and Security (MINES'09)*, pages 403–406. IEEE, 2009.
- [5] I. Kotera, R. Egawa, H. Takizawa, and H. Kobayashi. Modeling of cache access behavior based on zipf's law. In *Proceedings of the 9th workshop on MEMory performance: DEALing with Applications, systems and architecture (MEDEA '08)*, pages 9–15. ACM, 2008.
- [6] J. K. Mullin. A second look at bloom filters. *Commun. ACM*, 26(8):570–571, 1983.
- [7] P. Rodriguez, C. Spanner, and E. W. Biersack. Analysis of web caching architectures: hierarchical and distributed caching. *IEEE/ACM Trans. Netw.*, 9(4):404–418, 2001.
- [8] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2012.
- [9] Y. Zhu and H. Jiang. False rate analysis of bloom filter replicas in distributed systems. In *Proceedings of the 2006 International Conference on Parallel Processing (ICPP '06)*, pages 255–262. IEEE Computer Society, 2006.
- [10] Y. Zhu, H. Jiang, J. Wang, and F. Xian. Hba: Distributed metadata management for large cluster-based storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):750–763, 2008.