# Scalable algorithms for monitoring activity traces

Julien Pilourdault

Communauté
UNIVERSITÉ Grenoble Alpes

**THÈSE**

Pour obtenir le grade de

**DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE**

Spécialité : **Informatique**

Arrêté ministérial : 7 Août 2006

Présentée par

**Julien Pilourdault**

Thèse dirigée par **Sihem Amer-Yahia**
et co-encadrée par **Vincent Leroy**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

# Scalable Algorithms for Monitoring Activity Traces

Thèse soutenue publiquement le **28 septembre 2017**,
devant le jury composé de :

**Mme. Nadia Brauner**
Professeur à l'Université Grenoble Alpes, Présidente
**M. Patrick Valduriez**
Directeur de recherche INRIA, Rapporteur
**M. Donald Kossmann**
Professeur à ETH Zurich, Rapporteur
**Mme. Yanlei Diao**
Professeur à l'École Polytechnique, Examinatrice
**Mme. Sihem Amer-Yahia**
Directrice de recherche CNRS, Directrice de thèse
**M. Vincent Leroy**
Maître de conférences à l'Université Grenoble Alpes, Co-Encadrant de thèse

# Abstract

## Scalable Algorithms for Monitoring Activity Traces

**Keywords:** Monitoring; Temporal Data; Distributed Processing; Joins; Crowdsourcing; Task Assignment

In this thesis, we study scalable algorithms for monitoring activity traces. In several domains, monitoring is a key ability to extract value from data and improve a system. This thesis aims to design algorithms for monitoring two kinds of activity traces.

First, we investigate temporal data monitoring. We introduce a new kind of interval join, that features scoring functions reflecting the degree of satisfaction of temporal predicates. We study these joins in the context of batch processing: we formalize Ranked Temporal Join (RTJ), that combine collections of intervals and return the $k$ best results. We show how to exploit the nature of temporal predicates and the properties of their associated scored semantics to design *TKIJ*, an efficient query evaluation approach on a distributed Map-Reduce architecture. Our extensive experiments on synthetic and real datasets show that *TKIJ* outperforms state-of-the-art competitors and provides very good performance for n-ary RTJ queries on temporal data. We also propose a *preliminary study* to extend our work on *TKIJ* to stream processing.

Second, we investigate monitoring in crowdsourcing. We advocate the need to incorporate motivation in task assignment. We propose to study an adaptive approach, that captures workers' motivation during task completion and use it to revise task assignment accordingly across iterations. We study two variants of motivation-aware task assignment: *Individual Task Assignment* (Ita) and *Holistic Task Assignment* (Hta).

First, we investigate Ita, where we assign tasks to workers individually, one worker at a time. We model Ita and show it is NP-Hard. We design three task assignment strategies that exploit various objectives. Our live experiments study the impact of each strategy on overall performance. We find that different strategies prevail for different performance dimensions. In particular, the strategy that assigns random and relevant tasks offers the best task throughput and the strategy that assigns tasks that best match a worker's compromise between task diversity and task payment has the best outcome quality. Our experiments confirm the need for adaptive motivation-aware task assignment.

Then, we study Hta, where we assign tasks to all available workers, holistically. We model Hta and show it is both NP-Hard and MaxSNP-Hard. We develop efficient approximation algorithms with provable guarantees. We conduct offline experiments to verify the efficiency of our algorithms. We also conduct online experiments with real workers and compare our approach with various non-adaptive assignment strategies. We find that our approach offers the best compromise between performance dimensions thereby assessing the need for adaptability.

# Résumé

**Mots-clés:** Monitoring; Données Temporelles; Traitement Distribué; Jointures; Crowdsourcing; Affectation de Tâches

Dans cette thèse, nous étudions des algorithmes pour le monitoring des traces d'activité à grande échelle. Le monitoring est une aptitude clé dans plusieurs domaines, permettant d'extraire de la valeur des données ou d'améliorer les performances d'un système.

Nous explorons d'abord le monitoring de données temporelles. Nous présentons un nouveau type de jointure sur des intervalles, qui inclut des fonctions de score caractérisant le degré de satisfaction de prédicats temporels.

Nous étudions ces jointures dans le contexte du batch processing (traitement par lots). Nous formalisons la Ranked Temporal Join (RTJ), une jointure qui combine des collections d'intervalles et retourne les $k$ meilleurs résultats. Nous montrons comment exploiter les propriétés des prédicats temporels et de la sémantique de score associée afin de concevoir *TKIJ*, une méthode d'évaluation de requête distribuée basée sur Map-Reduce. Nos expériences sur des données synthétiques et réelles montrent que *TKIJ* est plus performant que les techniques de l'état de l'art et démontre de bonnes performances sur des requêtes RTJ $n$-aires sur des données temporelles. Nous proposons également une *étude préliminaire* afin d'étendre nos travaux sur *TKIJ* au domaine du stream processing (traitement de flots).

Nous explorons ensuite le monitoring dans le crowdsourcing (production participative). Nous soutenons la nécessité d'intégrer la motivation des travailleurs dans le processus d'affectation des tâches. Nous proposons d'étudier une approche adaptative, qui évalue la motivation des travailleurs lors de l'exécution des tâches et l'exploite afin d'améliorer l'affectation de tâches qui est réalisée de manière itérative.

Nous explorons une première variante nommée *Individual Task Assignment* (Ita), dans laquelle les tâches sont affectées individuellement, un travailleur à la fois. Nous modélisons Ita et montrons que ce problème est NP-Difficile. Nous proposons trois méthodes d'affectation de tâches qui poursuivent différents objectifs. Nos expériences en ligne étudient l'impact de chaque méthode sur la performance globale dans l'exécution de tâches. Nous observons que différentes stratégies sont dominantes sur les différentes dimensions de performance. En particulier, la méthode affectant des tâches aléatoires et correspondant aux intérêts d'un travailleur donne le meilleur flux d'exécution de tâches. La méthode affectant des tâches correspondant au compromis d'un travailleur entre diversité et niveau de rémunération des tâches donne le meilleur niveau de qualité. Nos expériences confirment l'utilité d'une affectation de tâches adaptative et tenant compte de la motivation.

Nous étudions une deuxième variante nommée *Holistic Task Assignment* (Hta), où les tâches sont affectées à tous les travailleurs disponibles, de manière holistique. Nous modélisons Hta et montrons que ce problème est NP-Difficile et MaxSNP-Difficile. Nous développons des algorithmes d'approximation pour Hta. Nous menons des expériences sur des données synthétiques pour évaluer l'efficacité de nos algorithmes. Nous conduisons également des expériences en ligne et comparons notre approche avec d'autres stratégies non adaptatives. Nous observons que notre approche présente le meilleur compromis sur les différentes dimensions de performance.

# Remerciements

Je remercie mes proches pour leur soutien lors de ces dernières années. Je suis également reconnaissant à Sihem Amer-Yahia et Vincent Leroy pour m'avoir permis de concrétiser cette thèse. Je remercie enfin toutes les autres personnes avec qui j'ai pu collaborer et qui m'ont aidé dans ce travail.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Considerable amounts of data of various types are continuously generated at increasing pace. This leads both academia and industry to pour many efforts into building systems that can handle such large datasets. These datasets are mostly known under the buzzword "Big Data", whose strict definition may vary [30, 75].

These datasets often have the form of *traces* of *activities*. Network traffic, physical activity, a discussion on Twitter or a visit on a website are some examples of activities. Firewall logs, measurement of physical sensors, tweets or web server logs are corresponding examples of traces. A number of systems generate activity traces.

In this thesis, we focus on a specific case of activity traces processing. We aim to build *algorithms* that enable efficient *monitoring* of activity traces. A general definition of monitoring is:

> The act of "observing and checking the progress or quality of (something) over a period of time" and/or "keeping under systematic review"[1]

Monitoring activity traces has applications in several domains. In this thesis, we focus on designing algorithms for monitoring two kinds of activity traces.

First, we focus on temporal data, that is often generated by sensors. For instance, in a data center, a system administrator has to monitor network traffic. She needs to query firewall logs to monitor potential threats. Such a query involves *connecting* data from several *collections* of client connections on her servers. These collections are usually large, since thousands of connections may be created during a single hour. She thus needs efficient algorithms to process these datasets. Moreover, such data usually includes timestamps, that denote when connections occurred, which is useful for traffic analysis. Concretely, the system administrator may want to find pairs of connections that happened simultaneously or forming a sequence. In another context, the quantified self domain, one may wish to monitor her performances. She would want to analyze speed and heartbeat measurements

[1]`https://en.oxforddictionaries.com/definition/monitor`

so as to discover if she has improved her performances. Because she may have different devices, she may also need to connect data from several collections.

The second kind of activities of interest are those generated by humans in the context of crowdsourcing. On crowdsourcing platforms, *workers* complete *tasks* published by requesters. A requester may need to improve crowdwork performance by adapting the assignment of tasks to workers. She naturally sees assignment as an iterative process, where each iteration is based on workers' performance during the previous iteration. Therefore, she has to monitor task completion and devise an adaptive assignment strategy that evolves over time.

In our two domains of investigation, monitoring is key to discovering valuable insights or improving a system's efficiency. This ability relies on algorithms that allow to *connect* data. For instance, in network traffic monitoring, connecting collections of client connections involves evaluating joins. Joins are both heavily used in various domains and very expensive to evaluate, hence various algorithms were developed to return efficiently join results. In crowdsourcing, task assignment relies on algorithms that connect workers to tasks and optimize various objectives. Because task assignment problems are often hard to solve, efficient algorithms — that often have the form of heuristics or approximations — were developed.

Since large amounts of activity traces are generated at higher paces, monitoring requires to tackle harder challenges. Systems must remain scalable and responsive to handle large datasets. This leads to the growing need to design *scalable* algorithms.

The goal of this thesis is to develop *scalable algorithms for monitoring temporal data and human activity traces in crowdsourcing.* The remainder of this chapter is organized as follows:

1. We introduce monitoring of temporal data in Section 1.1. Specifically, we focus on processing joins on interval data.

2. We introduce adaptive task assignment in crowdsourcing in Section 1.2. Here, monitoring is interpreted as continuous task assignment and capturing workers' motivation.

3. We present an overview of this thesis and of our contributions in Section 1.3.

## 1.1 Monitoring Temporal Data

The definition of monitoring features the notion of evolution over time. Therefore, it is natural to find applications of monitoring on temporal datasets. We present here examples of temporal data and focus on the commonly used interval data.

## 1.1.1 Temporal Data and Temporal Joins

Because a number of systems generate records that contain timestamps, temporal data is pervasive. Examples include store receipts, webserver logs, or temperature measures generated by weather sensors or by wearables. Such data is often best represented as intervals with start and end timestamps. We illustrate two kinds of temporal data that can be represented as intervals in the examples below.

*Example* (Network Traffic Monitoring). In a data center, network traffic generates continuously temporal data. Specifically, firewall logs collect large amounts of traffic packets exchanged between servers and clients. Each packet has a timestamp. The *connection* of a client on a server may be represented by grouping close consecutive packets exchanged between a (client, server) pair. Therefore, a connection has the form [*IP_ client*,*IP_ server*, *start timestamp*,*end timestamp*,...] and represents the activity of the client on the server between two timestamps.

*Example* (Tweet Analysis). Tweets are associated to hashtags that characterize their topics. An analyst may be interested in querying trending topics. To do so, she defines the lifespan of a hashtag as the period during which it was popular (e.g. it was used more than 1000 times per hour). She obtains an interval [*hashtag*, *start time popular period*, *end time popular period*] that represents the lifetime of a trending topic.

**Temporal Joins.** Monitoring temporal data involves connecting data that comes from several collections. Such an operation is usually expressed using *joins*. In this thesis, we focus on *interval joins*, that connect interval data from several *collections* using temporal *relations*. Temporal relations between intervals are typically expressed using the Allen interval algebra [7], which defines a set of Boolean *predicates* such as *before, meets, starts* and *overlaps*. For instance, *before*$(x, y)$ requires that $x$ ends before $y$ starts and *meets*$(x, y)$ requires that $x$ ends exactly when $y$ starts. We revisit the previous example to illustrate interval joins.

*Example* (Network Traffic Monitoring). A data center administrator wishes to monitor traffic emanating from two countries. She has two collections of connections, one for each country. She formulates a query that returns pairs of connections $(x, y)$ where *before*$(x, y)$ and $x$ and $y$ originate from different countries.

*Example* (Tweet analysis). An analyst has a collection of trending topics that characterize notorious events. She is interested in detecting causality between those events. She would need to find pairs of discussion topics where one started when the other ended. Thus, she would need pairs of topics $(x, y)$ where *meets*$(x, y)$.

Processing interval joins has been extensively studied in both centralized [35, 44, 54] and parallel settings [29, 113]. For instance, the queries presented above can be efficiently evaluated in a distributed setting using Map-Reduce [29].

### 1.1.2 Top-k Temporal Joins: Problem and Challenges

We showed examples where temporal joins featuring a Boolean interpretation of temporal predicates help monitoring activity traces. In this thesis, we argue that a wider range of scenarios should be considered. The ability to evaluate interval predicates *approximately* and assign scores to resulting interval pairs, appears as a natural requirement to finding interesting results. In practice, joins need to be evaluated approximately to cope with imprecise measurements and the need to cover more results. The following revisited examples illustrate this idea.

*Example* (Network Traffic Monitoring). A pair of two connections that occurred at very different periods may not be relevant since they may not be related. Therefore, a connection pair $(x, y)$ where $x$ ends *just before* $y$ may be preferred to those where $x$ ends *much earlier than* the start of $y$.

*Example* (Tweets Analysis). A Boolean interpretation of $x$ *meets* $y$ is likely to return an empty result or very few results and will not detect discussion topics that started *roughly* after another ended. A small delay may indeed be acceptable.

**Problem.** To the best of our knowledge, no previous study has investigated approximate interval joins. It is crucial to (i) formalize a query model that enables using approximate predicates and (ii) define an efficient way to evaluate join queries that use those predicates. Our problem is thus *how to formalize and evaluate efficiently a temporal join query that features an approximate interpretation of temporal predicates?*

**Challenges.** Our problem incurs new challenges. First, we need to devise an appropriate semantics for a variety of temporal predicates in order to capture the strength of relationship between two intervals. This includes the formalization of a new kind of query. Second, we need to design an efficient query evaluation approach: evaluating joins is known to be expensive and an analyst needs a decent response time with a scalable approach when querying large datasets.

To the best of our knowledge, existing techniques are not applicable to our settings. First, we aim to use an approximate interpretation of temporal predicates whereas previous studies rely on a Boolean interpretation of Allen's predicates [29] or the *intersects* predicate [35, 44, 88], that is verified only if two intervals have a non-empty intersection. Second, traditional techniques on *top-k* or *rank-join* processing [45, 46, 51, 72, 95] rely on sorting or indexing objects using their scores. In our settings, the strength of relationship between two intervals can also be seen as a score. However, this score is unknown *a priori*, since it depends on pairs of intervals, that need to be materialized to compute scores. This makes traditional techniques inapplicable.

### 1.1.3 Scope: Batch and Stream Processing

We study two processing paradigms for temporal data monitoring. First, we investigate algorithms for **batch processing** of temporal data. In this setting, all data is given at once. For instance, in network traffic monitoring, an analyst queries the log of a whole day after all data has been collected. Here, monitoring is processed *offline.*

We also propose a *preliminary study* for extending our investigations on **stream processing**. Here, data arrives in the form of streams that are a "real-time, continuous, ordered sequence of items" [57]. For instance, in network traffic monitoring, an analyst may want to query client connections as they arrive, or just a few minutes after they have finished. Here, monitoring happens in *real-time* or *online.*

## 1.2 Monitoring in Crowdsourcing

Our second domain of investigation is crowdsourcing. Here, we focus on the activity of *workers* who complete *tasks.* We first introduce the concept of crowdsourcing, then, we argue why monitoring plays a key role in virtual marketplaces.

### 1.2.1 Crowdsourcing and Adaptive Task Assignment

The term "crowdsourcing" (a neologism coming from *outsourcing*) was originally defined by Jeff Howe[2][70] as

> "the act of a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call"

This definition encompasses several activities such as collaborative tasks, project work, surveys or microtasking. Crowdsourcing has notably emerged as an efficient way to complete tasks that are easier for humans, but harder for computers. The example below illustrates collaborative crowdsourcing in a science project.

*Example* (Collaborative Crowdsourcing). Galaxy Zoo[3] is a "citizen science project". In 2007, researchers published a number of galaxies images and invited visitors to classify these galaxies according to their shape. This allowed to get more than 50M classifications in a year and to overpass difficulties raised by other classical solutions (i.e. having a few professional astronomers working on thousands of images or using computers programs that were not accurate enough).

[2] `http://www.crowdsourcing.com/cs/2006/06/crowdsourcing_a.html`
[3] `https://www.galaxyzoo.org`

Other popular examples include writing a Wikipedia article — that can be seen as a collaborative task, collaborative creation of maps (OpenStreetMap[4], Crowdmap[5]) or transcription of audio files for a public library (NYPL Labs[6]).

Crowdsourcing is not only about volunteering. A number of online crowdsourcing *platforms* have been developed[7] to enable a professionalization of crowdsourcing. A popular example is Amazon Mechanical Turk (AMT), which is a *micro-tasking* platform. On AMT, *requesters* publish *HITs*, that are essentially *micro-tasks*. Then, workers complete these HITs and submit their work to get a financial reward. A micro-task usually includes few questions that can be completed in less than a minute. Tagging an image or classification tasks are popular examples of micro-tasks published on AMT [74]. The example below is a concrete illustration of a micro-task that may be published on a crowdsourcing platform.

*Example* (Micro-Task). An airline company `AirWorld` has collected tweets that feature `#AirWorld` as hashtag. The customer service aims to get the opinion of people about the company. They choose to publish tasks on a micro-tasking platform. In each task, a worker is asked to classify the tone of a tweet (positive, negative, neutral) and if it is negative, give the reason of dissatisfaction. Once all tweets have been classified, the customer service can get a precise insight of people's opinion about `AirWorld`.

**Adaptive Task Assignment.** On most crowdsourcing platforms, requesters rely on *self-appointment* of workers to tasks. Workers look for tasks that they prefer using filters or ranking proposed on the platform. This approach has several caveats. For instance, workers waste time looking for interesting tasks, while they could be paid to complete tasks. Moreover, requesters may need to wait longer to get published tasks completed. Additionally, crowdwork quality may not be sufficient as inexperienced workers may complete tasks. Several techniques are employed to tackle this latter problem such as requiring qualifications for workers or imposing them to complete test tasks (also known as *golden* tasks) to evaluate them. However, they may be tedious to design or have limitations in complex tasks that require knowledge [47]. To tackle this problem, multiple research efforts were driven towards *algorithmic task assignment* [47, 55, 67, 68, 106, 109, 135]. Here, tasks are assigned to workers using some algorithm. Task assignment usually considers goals such as maximizing quality of the crowdwork or meeting a deadline. A number of studies focused on *adaptive* task assignment [47, 67, 68, 135] where past crowdwork is leveraged to improve task assignment. For instance, one may learn worker's skills based on the tasks that she completed before and assign her task that best match her skills.

Interestingly, adaptive task assignment can be interpreted as the monitoring of workers' activity. Indeed, it requires to *observe* workers and *check* how tasks are completed to

[4]`http://www.openstreetmap.org`

[5]`https://crowdmap.com`

[6]`http://togetherwelisten.nypl.org`

[7]`http://www.mturk.com`, `https://www.foulefactory.com`, `https://www.crowdflower.com`

improve task assignment. Therefore, monitoring is crucial in crowdsourcing, since it helps to tackle important challenges such as improving crowdwork quality.

### 1.2.2 Motivation-Aware Task Assignment: Problem and Challenges

Task assignment has received much attention [47, 55, 67, 68, 106, 109, 135]. Since the 70's, on the other hand, organization studies explored worker motivation in physical workplaces [62], and recent work has shown that a similar motivation model prevails in virtual marketplaces [79, 108]. In particular, worker motivation has been shown to play a key role in task completion [25, 33]. In this thesis, we combine these two ideas and investigate motivation-aware micro-task assignment based on the following hypothesis: *Since workers' motivation evolves as they complete tasks, tasks should be re-assigned to workers accordingly, in order to improve the quality of their contributions.* To the best of our knowledge, existing work on crowdsourcing does not incorporate motivation in task assignment. Instead, motivation is treated as an external factor in task completion, by incentivizing workers for long-lasting tasks [25, 69], or entertaining them when they complete several tasks [33]. We argue that *a principled way of leveraging motivation is to incorporate it directly in the task assignment process*, i.e., in choosing which tasks to assign to which workers.

**Problem.** We define our problem as *how to monitor workers and incorporate motivation in task assignment ?* We believe solving this problem will lead to *high quality contributions*, and improve *overall performance.*

**Challenges.** Our first challenge is to model a worker's motivation. Some workers may be driven by fun and enjoyment, others may look to advance their human capital, or increase their compensation [79]. We need to devise a model that captures accurately a worker's motivation. Our second challenge is to formulate motivation-aware task assignment. Task assignment has been formulated as a one-shot optimization problem whereby goals such as maximizing task quality, or minimizing cost and latency, are used when matching tasks and workers [106, 109]. The difficulty with worker motivation is that *it must be re-captured as workers complete more tasks, and tasks must be assigned to them accordingly and adaptively.* Our third challenge is to design efficient assignment algorithms. The crowd is volatile and usually not well-engaged: it is crucial that the response time of our algorithms is optimized so that workers are never waiting for being assigned motivating tasks.

### 1.2.3 Scope: Individual and Holistic Task Assignment

We study two natural variants of motivation-aware task assignment. The first one, **Individual Task Assignment** (Ita), *individually* assigns tasks to *one worker at a time.* The second one, **Holistic Task Assignment** (Hta), assigns tasks to *all available workers, holistically.*

## 1.3 Overview of this thesis and contributions

This thesis is organized around its two main domains of investigation. We pursue the following outline:

1. In Chapter 2, we present our investigations on monitoring temporal data.

   (a) We present our proposal for a new kind of join for monitoring temporal data.

   (b) We model approximate temporal predicates, that are an adaptation of a flexible scoring approach for Allen's predicates.

   (c) We study the efficient evaluation of temporal join queries in distributed batch processing [103]:

      1. We study how to formalize temporal join queries that feature approximate predicates. We introduce *n*-ary RTJ queries.
      2. We study how to design an efficient query evaluation approach for RTJ queries. We design *TKIJ*, a query evaluation approach on Map-Reduce that leverages scores to avoid computing unnecessary results and to balance the load between reducers.
      3. We conduct extensive experiments on synthetic and real datasets showing the very good performance of *TKIJ* for n-ary RTJ queries on temporal data.

   (d) We propose a *preliminary study* to extend our contribution to stream processing. We formalize a query model and propose investigation directions to design an efficient query evaluation approach in the context of stream processing.

2. In Chapter 3, we study the problem of motivation-aware task assignment.

   (a) We present our proposal for motivation-aware task assignment and why we advocate an adaptive task assignment approach.

   (b) We present our model for motivation-aware task assignment. First, we model tasks and workers. Then, we formalize motivation factors and show how we capture the preference of a worker for a given factor.

   (c) We study *Individual Task Assignment* (Ita) [101]:

      i. We model the Ita problem and show it is NP-Hard. We define motivation as a combination of two factors: *task diversity* and *task payment.*

      ii. We design three assignment strategies for Ita that exploit different objectives: RELEVANCE, DIV-PAY and DIVERSITY.

      iii. We conduct live experiments with real workers to evaluate these strategies showing that DIV-PAY, an adaptive motivation-aware strategy, leads to higher quality contribution.

(d) We study *Holistic Task Assignment* (Hta) [102].

i. We model the Hta problem and show it is NP-Hard and also MaxSNP-Hard. We consider a different definition of motivation, that combines *task diversity* and *task relevance.*

ii. We develop HTA-APP and HTA-GRE, two approximation algorithms for Hta. HTA-APP has a better approximation factor ($\frac{1}{4}$) than HTA-GRE ($\frac{1}{8}$), with the cost of a higher running time.

iii. We conduct synthetic experiments with real tasks showing the superiority of HTA-GRE over HTA-APP: the higher running time of HTA-APP does not yield better results for Hta.

iv. We conduct live experiments with real workers to evaluate HTA-APP against several alternatives. We find that HTA-GRE offers a good compromise between various performance dimensions: quality, number of completed tasks and worker retention.

3. We conclude in Chapter 4, where we propose future directions for our investigations.

## 1.4 Applications

In this thesis, we assessed the relevance of our investigations in two concrete application domains.

**Network Traffic Monitoring.** We contributed to the DATALYSE project[8], a consortium involving computer science laboratories and companies. Specifically, we collaborated with a data hosting company that owns datacenters. We worked on concrete usecases to help monitoring datacenters and on shipping our implementation of *TKIJ*. Concretely, *TKIJ* is a 3-phase Map-Reduce algorithm that counts $\approx 11,000$ lines of Java code.

This collaboration allowed us to run experiments on real data. Specifically, we use network traffic data collected on firewall logs. For instance, we use packet traces, that represent activities of client on servers. We employ files that include all packet traces for a single day. We transform these traces to process joins on interval collections that include $\approx$ 3M intervals, which is in the same order of magnitude as collections employed in experiments for state-of-the-art competitors [29]. We verify the efficiency of our algorithm on various queries that are useful to monitor network traffic.

**Adaptive Crowdsourcing.** Our work includes the design and the implementation of a crowdsourcing platform, coined GACS (Grenoble Adaptive CrowdSourcing). Existing platforms, such as AMT, do not enable adaptive task assignment although they benefit

[8]http://www.datalyse.fr

from a large workforce. Therefore, we chose to hire workers from such platforms and redirect them to GACS. Concretely, GACS is a $\approx 2,800$ lines of code JEE web application backed by a Postgresql database. It can be interfaced with various assignment strategies or data models.

We employed GACS to conduct experiments with real workers hired from AMT. Workers were assigned real tasks that we extracted from Crowdflower, a popular crowdsourcing platform. We conducted two main experiments, where up to 58 different workers completed $2,715$ tasks in 80 work sessions. These experiments allowed us to assess the relevance of our approaches in a concrete setting.

# Chapter 2

# Monitoring Temporal Data

We exposed in Chapter 1 how joins on temporal data could be interpreted as monitoring activity traces. We sketched the problem of monitoring temporal data using an approximate interpretation of predicates. In this chapter, we present our investigations on temporal data monitoring. We propose to study a new kind of join supporting an approximate interpretation of temporal predicates and returning only best results. Then, we propose an efficient query evaluation approach on Map-Reduce. Finally, we study an extension of our work for stream processing.

## 2.1 Our Proposal: Top-k Temporal Joins

Joins have a key role in a number of applications. We illustrate in the following example how joins may be used for monitoring network traffic data.

*Example* (Network Traffic Monitoring). In a data center, a tuple (*IP_ client*, *IP_ server*, *start timestamp*, *end timestamp*, …) represents a *connection* of a client on a server. The interval [ *IP_ client*,*IP_ server*] represents the lifetime of the client activity on the server. A system administrator has to monitor the data center which requires to monitor traffic emanating from different countries. The firewall has collected two logs of that have the form of collections of connections, one for each country. She needs to find abrupt changes of traffic between two countries, and she has two collections $C_1$ and $C_2$ as exposed in Figure 2.1. She would formulate a query that returns pairs of requests $(x, y) \in C_1 \times C_2$ where $x$ starts as $y$ ends or equivalently $x$ *meets* $y$. Such pairs may be interesting since they may reflect some causality between two connections. The best sequences are those satisfying *meets*$(x, y)$ reflecting strict equality between the end of $x$ and the beginning of $y$. In our example, only $(x_4, y_4)$ qualifies for a Boolean interpretation.

Temporal relations such as *meets* are typically expressed using the popular Allen interval algebra [7]. Allen defined a set of Boolean predicates such as *before*, *meets* and *overlaps* that characterize relations between intervals. In this thesis, we argue that the ability to

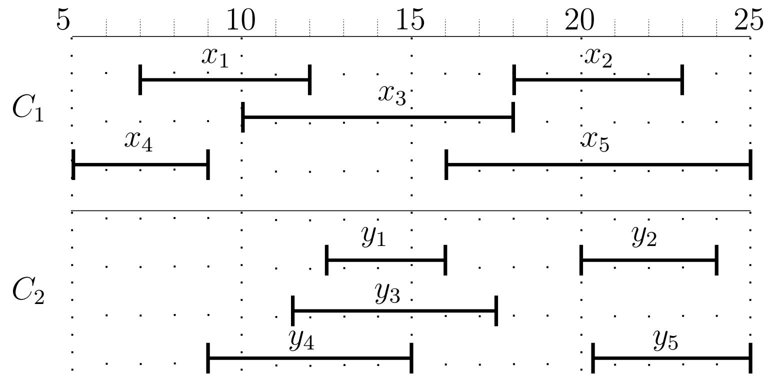


Figure 2.1: Motivating example for top-$k$ temporal joins

evaluate interval predicates *approximately* and assign scores to resulting interval pairs, appears as a natural requirement to finding interesting results. Moreover, a query should return only *best* results, which can be interpreted as returning only the *top-k* results. We illustrate this idea in the example below.

*Example* (Network Traffic Monitoring, continued). In our example, only $(x_4, y_4)$ is returned if a Boolean interpretation is employed. Yet, given the uncertainty on clocks in different network equipments, a small overlap or a short delay between two consecutive connections would be more realistic. In order to express that with Boolean semantics, a query must evaluate a disjunction of *before*, *overlaps* and *meets* and would build many useless interval pairs. A ranked semantics on the other hand, would build $(x, y)$ pairs where $x$ *almost meets* $y$, allowing a tolerance on intervals' endpoints. For instance, tuples $(x, y)$ where $x$ ends at most 2 timestamps before or after $y$ starts, could be considered high-scoring. Using such a semantics, we can return the top-3 $\{(x_4, y_4), (x_1, y_3), (x_1, y_1)\}$.

## Proposal

Our example shows the need to support a wider range of scenarios. We propose to formalize n-ary Ranked Temporal Join (RTJ) queries that feature *approximate* temporal predicates and return the $k$ best results.

## Scope

First, we investigate algorithms for **batch processing** of temporal data. We assume that we are given interval *collections*. For instance, in network traffic monitoring, an analyst queries the log of a whole day after all data has been collected. In tweet analysis, all tweets over a given period may be collected before querying the dataset. In this context, the query evaluation response time is usually in *seconds* or *minutes* and data is processed *offline*. This enables some optimizations for query evaluation. For instance, we can pre-process datasets (e.g. collect statistics) to optimize join processing. Our investigation includes the design of an efficient query evaluation approach on Map-Reduce [34], a popular framework for distributed batch processing of large datasets.

Second, we conduct a *preliminary study* for extending our investigations on **stream processing**. In this setting, data arrives in the form of streams that are a “real-time, continuous, ordered sequence of items” [57]. Streams are usually “multiple, rapid, time-varying, [and] possibly unpredictable” [12] which raises new challenges [57, 119]. For instance, in network traffic monitoring, an analyst may want to query client connections as they arrive, or just a few minutes after they have finished. Here, monitoring happens in *real-time* or *online* and we expect a query evaluation response time in *sub-seconds* or *seconds*. We aim to design a distributed query evaluation approach, that could be implemented on Apache Storm [121], a popular framework for distributed stream processing.

### 2.1.1 Top-k Temporal Joins: Challenges

RTJ queries raise a number of new challenges. First, an appropriate ranked semantics needs to be devised for a variety of temporal predicates in order to capture the strength of relationship between intervals as a function of the desired semantics for each predicate. Most related studies focus on a common query that involves the *intersects* predicate [35, 44, 88]. The idea is to retrieve tuples that share a common period of validity, in order to combine events whose time range has a non-empty intersection. Our semantics is richer since we are interested in any temporal predicate between intervals. For instance, we aim to capture predicates from the Allen algebra (see the three first columns of Figure 2.2).

It is important to note that, unlike existing work on returning approximate answers [20], we focus on computing *exact* answers of queries that make use of scored join predicates. The second challenge we tackle is to devise an efficient query evaluation strategy that guarantees to return the *best* answers for a variety of temporal predicates. The key difficulty here is to develop a general-purpose algorithm that works with *a variety of predicates and ranked semantics* and yet, that is able to exploit the nature of those predicates to devise an efficient evaluation.

### 2.1.2 Overview of our Contributions

In this chapter, we present our investigations on temporal data monitoring:

(a) We study how to model approximate temporal predicates. We need a model that supports a range of queries on temporal data. We aim to support queries shown in our concrete examples in network traffic monitoring or tweet analysis. We propose to adapt a flexible scoring approach for Allen’s predicates. We present temporal predicates in Section 2.2.

(b) We study batch processing of top-$k$ temporal joins in Section 2.3 [103]:

1. We formalize $n$-ary Ranked Temporal Join queries (RTJ), that combine interval collections with any number of approximate interval predicates and return the $k$ best results. We introduce $n$-ary RTJ queries in Section 2.3.1.
2. We design *TKIJ*, an efficient RTJ query evaluation approach on a distributed Map-Reduce architecture. *TKIJ* exploits the nature of temporal predicates to prune unnecessary results and optimize workload assignment to reducers. We present *TKIJ* in Section 2.3.2.
3. We run extensive experiments to evaluate *TKIJ* on synthetic and real network traffic datasets. Our experiments show the efficiency of our pruning technique and the great benefit of using scores to distribute the load between reducers. Because *TKIJ* executes only combinations that ensure to return the correct top-$k$ answers, it scales to very large collections (up to 5M tuples per collection) and to high $k$ values (up to $10^5$). Section 2.3.3 presents our experiments.

(c) We propose a *preliminary study* for extending RTJ queries to stream processing in Section 2.4. We outline main components for monitoring temporal data using stream processing:

   (a) We formalize S-RTJ queries, an extension of RTJ queries to stream processing.

   (b) We propose investigation directions to design an efficient distributed evaluation approach for S-RTJ queries.

## 2.2 Temporal Predicates

A key observation we rely on is that an approximate interpretation of a predicate amounts to *approximating the strength of relationship between its intervals' endpoints*. That is compatible with the flexible scoring approach proposed by Dubois et al. [41] to approximate Allen predicates [7]. It is based on associating a degree of satisfaction with equality and inequality of 2 intervals' endpoints. In this thesis, we adapt this framework to allow scoring any temporal predicate. We also allow to use any monotone aggregation function to compute the score of a query result for our queries. Consequently, a 3-way query involving the predicates $\mathit{starts}(x, y)$ and $\mathit{meets}(y, z)$ would return $(x, y, z)$ tuples and associate to each tuple its degree of satisfaction of the query as an aggregation of individual predicate-dependent scores for each of $\mathit{starts}(x, y)$ and $\mathit{meets}(y, z)$.

**Boolean temporal predicates.** The general form of a temporal predicate between two intervals $x$ and $y$ is denoted $p(x, y)$ and is expressed as a Boolean conjunction of equalities and inequalities between their endpoints $\underline{x}, \overline{x}, \underline{y}, \overline{y}$. This allows to capture a wide range of predicates among which the seminal Allen algebra [7]. The first 3 columns of Figure 2.2

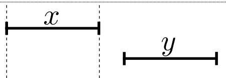
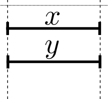
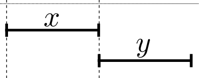
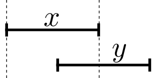
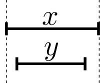
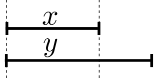
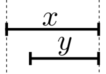
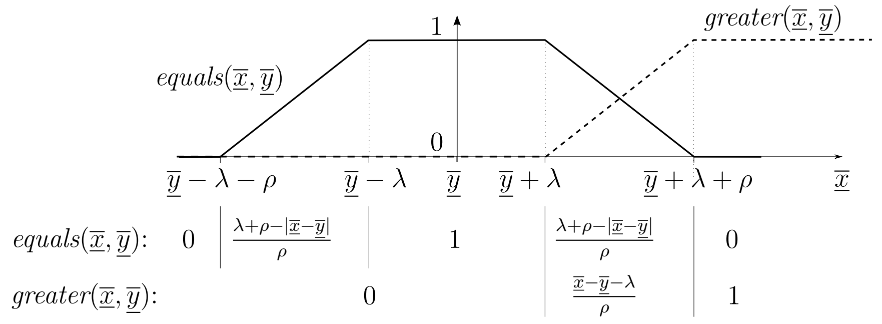
| **Temporal Predicate** | **Boolean Interpretation** | **Valid answers** | **Scored Intepretation** |
|---|---|---|---|
| $before(x, y)$ | $\overline{x} < \underline{y}$ |  | $\textbf{s-before}(x, y) = greater(\underline{y}, \overline{x})$ |
| $equals(x, y)$ | $\underline{x} = \underline{y} \wedge \overline{x} = \overline{y}$ |  | $\textbf{s-equals}(x, y) = \min\{equals(\underline{x}, \underline{y}), equals(\overline{x}, \overline{y})\}$ |
| $meets(x, y)$ | $\overline{x} = \underline{y}$ |  | $\textbf{s-meets}(x, y) = equals(\overline{x}, \underline{y})$ |
| $overlaps(x, y)$ | $\underline{x} < \underline{y} \wedge \overline{x} > \underline{y} \wedge \overline{x} < \overline{y}$ |  | $\textbf{s-overlaps}(x, y) = \min\{greater(\underline{y}, \underline{x}), greater(\overline{x}, \underline{y}), greater(\overline{y}, \overline{x})\}$ |
| $contains(x, y)$ | $\underline{x} < \underline{y} \wedge \overline{x} > \overline{y}$ |  | $\textbf{s-contains}(x, y) = \min\{greater(\underline{y}, \underline{x}), greater(\overline{x}, \overline{y})\}$ |
| $starts(x, y)$ | $\underline{x} = \underline{y} \wedge \overline{x} < \overline{y}$ |  | $\textbf{s-starts}(x, y) = \min\{equals(\underline{x}, \underline{y}), greater(\overline{y}, \overline{x})\}$ |
| $finishedBy(x, y)$ | $\underline{x} < \underline{y} \wedge \overline{x} = \overline{y}$ |  | $\textbf{s-finishedBy}(x, y) = \min\{greater(\underline{y}, \underline{x}), equals(\overline{x}, \overline{y})\}$ |

Figure 2.2: The Allen algebra with Boolean and scored temporal predicates.

summarize Allen temporal predicates and their semantics. For example, $meets(x, y)$ imposes that $y$ starts when $x$ finishes while $starts(x, y)$ requires that $x$ and $y$ start at the same time and that $x$ ends before $y$.

It is important to note that we aim to capture all Allen predicates but also any predicate comparing interval's endpoints. While we do not aim to provide a long list of new predicates, we discuss examples that we are using in our experiments. For example, in network traffic analysis, we introduce $justBefore(x, y)$ that is satisfied iff $\underline{y} > \overline{x}$ and $\underline{y} - \overline{x} \leq AVG_z(\overline{z} - \underline{z})$. The intuition is that the elapsed time between $x$ and $y$ is no greater than the average interval length. A special case is the predicate $shiftMeets(x, y)$ that is satisfied iff $(\underline{y} - \overline{x}) = AVG_z(\overline{z} - \underline{z})$. In tweet analysis, a possibly useful predicate would be $sparks(x, y)$ which is satisfied iff $(\overline{y} - \underline{y}) > 10 * (\overline{x} - \underline{x})$ and $\underline{y} > \overline{x}$. As a result, the $(x, y)$ pairs satisfying $sparks(x, y)$ would identify hashtag pairs where the preceding hashtag lasted 10 times shorter that the following. That could be useful in determining causality of long-lasting events such as finding all short-lasting hashtags before the long-lasting #`JeSuisCharlie`.

**Scored temporal predicates.** Since we are interested in capturing the degree at which a temporal predicate is verified by a pair of intervals, we propose to associate a score



1
0
$equals(\underline{\overline{x}}, \underline{\overline{y}})$
$greater(\underline{\overline{x}}, \underline{\overline{y}})$
$\underline{\overline{y}} - \lambda - \rho$
$\underline{\overline{y}} - \lambda$
$\underline{\overline{y}}$
$\underline{\overline{y}} + \lambda$
$\underline{\overline{y}} + \lambda + \rho$
$\underline{\overline{x}}$
$equals(\underline{\overline{x}}, \underline{\overline{y}})$: 0 | $\frac{\lambda+\rho-|\underline{\overline{x}}-\underline{\overline{y}}|}{\rho}$ | 1 | $\frac{\lambda+\rho-|\underline{\overline{x}}-\underline{\overline{y}}|}{\rho}$ | 0
$greater(\underline{\overline{x}}, \underline{\overline{y}})$: 0 | $\frac{\underline{\overline{x}}-\underline{\overline{y}}-\lambda}{\rho}$ | 1

Figure 2.3: Approximating *equals* and *greater*. Here, $\underline{\overline{x}} \in \{\underline{x}, \overline{x}\}$, $\underline{\overline{y}} \in \{\underline{y}, \overline{y}\}$.

to each predicate. Here again, we aim to be general and we adopt the flexible approach for scoring Allen predicates [41] and adapt it to our settings. This approach relies on two primitive approximation comparators on intervals' endpoints. Those comparators, $equals(\underline{\overline{x}}, \underline{\overline{y}})$ and $greater(\underline{\overline{x}}, \underline{\overline{y}})$, are used to express the degree of equality or inequality of intervals' endpoints $\underline{\overline{x}}$ and $\underline{\overline{y}}$, where $\underline{\overline{x}} \in \{\underline{x}, \overline{x}\}$, $\underline{\overline{y}} \in \{\underline{y}, \overline{y}\}$ as a graded value in $[0, 1]$. They rely on two parameters $\lambda$ and $\rho$ that provide flexibility in controlling the tolerance degree when comparing intervals' endpoints. Figure 2.3 shows how $equals(\underline{\overline{x}}, \underline{\overline{y}})$ and $greater(\underline{\overline{x}}, \underline{\overline{y}})$ are used with $\lambda$ and $\rho$ to express that tolerance. For instance, by defining that whenever $|\underline{\overline{x}} - \underline{\overline{y}}| \leq \lambda$, $equals(\underline{\overline{x}}, \underline{\overline{y}})$ returns 1, $\lambda$ sets a tolerance for exact endpoint equality. $\rho$, on the other hand, is used to define score values. A large $\rho$ value defines a wide range of score values and a small $\rho$ produces a more abrupt curve and fewer possible score values.

Since temporal predicates are expressed as equalities and inequalities on intervals' endpoints, their approximation can be achieved using a conjunction of $equals()$ and $greater()$ with appropriate $\lambda$ and $\rho$ values. This allows us to *associate a scored variant to each temporal predicate.* We denote that variant $s\text{-}p(x, y)$ and refer to it as *scored temporal predicate*, abusing the term "predicate" to mean "function". Indeed, while $p(x, y)$ returns a Boolean value, $s\text{-}p(x, y)$, returns a score in [0,1]. For example, we can define the scored version of $starts(x, y)$ as $s\text{-}starts(x, y) = \min\{equals(\underline{x}, \underline{y}), greater(\overline{y}, \overline{x})\}$.

We also propose to allow different values of $\lambda$ and $\rho$ for $equals()$ and $greater()$ for different predicates. That provides a finer control of the score values produced by each predicate. A Boolean interpretation of a predicate becomes a special case of our scored interpretation. For example, strict endpoint equality can be obtained by setting both $\lambda_{equals}$ and $\rho_{equals}$ to 0. Thus, we can define, $s\text{-}justBefore(x, y)$ with $\lambda_{greater}$ and $\rho_{greater}$ set to 0, $\rho_{equals}$ to any value and $\lambda_{equals}$ to $AVG_z(\overline{z} - \underline{z})$ (Figure 2.4).

## 2.3 Top-k Temporal Joins: Batch Processing

In this section, we present our investigations for monitoring temporal data using batch processing. The efficient batch processing of interval joins has been studied before [29,

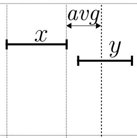
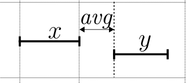
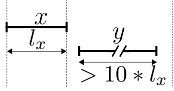
| **Temporal Predicate** | **Boolean Interpretation** | **Valid Answers** | **Scored Intepretation** |
|---|---|---|---|
| *justBefore*$(x, y)$ | $\overline{x} < \underline{y} \wedge$ $\underline{y} - \overline{x} \leq avg$ |  | $\boldsymbol{s\text{-}justBefore}(x, y) =$ $\min\{equals(\overline{x}, \underline{y}),$ $greater(\underline{y}, \overline{x})\}$ $\lambda_{greater} = \rho_{greater} = 0,$ $\lambda_{equals} = avg,\ \rho_{equals} \in \mathbb{R}^+$ |
| *shiftMeets*$(x, y)$ | $\underline{y} = \overline{x} + avg$ |  | $\boldsymbol{s\text{-}shiftMeets}(x, y) =$ $equals(\overline{x} + avg, \underline{y})$ |
| *sparks*$(x, y)$ | $\overline{x} < \underline{y} \wedge$ $(\overline{y} - \underline{y}) >$ $10 * (\overline{x} - \underline{x})$ |  | $\boldsymbol{s\text{-}sparks}(x, y) =$ $\min\{greater(\underline{y}, \overline{x}),$ $greater(\overline{y} - \underline{y}, 10 * (\overline{x} - \underline{x}))\}$ |

Figure 2.4: Definition of *s-shiftMeets*, *s-justBefore* and *s-sparks*. Here, $avg = AVG_z(\overline{z} - \underline{z})$.

35, 44, 88]. The closest to our work is the recent one by Chawda et al. [29] with a focus on processing queries on Map-Reduce [34]. However, their algorithms focus on a Boolean semantics and are not directly applicable in our case. In our work, scores constitute both a challenge and an opportunity. They are a challenge because, unlike Boolean semantics, every combination of intervals is potentially an answer. The opportunity lies in the ability to leverage statistics on input data in order to avoid computing low-scoring results.

We present our investigations as follows. First, we model RTJ queries, that include an approximate interpretation of temporal predicates. Second, we propose *TKIJ*, a 3-stage query evaluation approach for RTJ queries on Map-Reduce. Third, we conduct extensive experiments to evaluate *TKIJ* on both synthetic and real network traffic datasets.

### 2.3.1 Data Model and Problem for Batch Temporal Joins

We are given $m$ collections of intervals $C_1, \ldots, C_m$. Each interval $x$ has a unique identifier, a start time $\underline{x}$ and an end time $\overline{x}$.

#### 2.3.1.1 Temporal join queries

We are interested in expressing n-ary join queries on interval collections $C_1, \ldots, C_m$. We express a query $Q$ as a weakly connected oriented simple graph[1] of the form $(V, E)$. Each each vertex $v_i \in V$ is mapped to a collection $C_i$. Each edge $(i, j) \in E$ between two vertices $v_i$ and $v_j$ is labeled with a scored temporal predicate $s\text{-}p_{(i,j)}()$ between the two collections $C_i$ and $C_j$ corresponding to $v_i$ and $v_j$.

The evaluation of an n-ary join query $Q$ returns a set of tuples of the form $(x_1, \ldots, x_n)$ where $x_i \in C_i$. The score of each tuple in the query result is computed using a function $\mathcal{S}$ that aggregates the partial scores assigned by each predicate $s\text{-}p_{(i,j)}()$ associated with each

[1]no self loops and $(i, j) \in E \implies (j, i) \notin E$

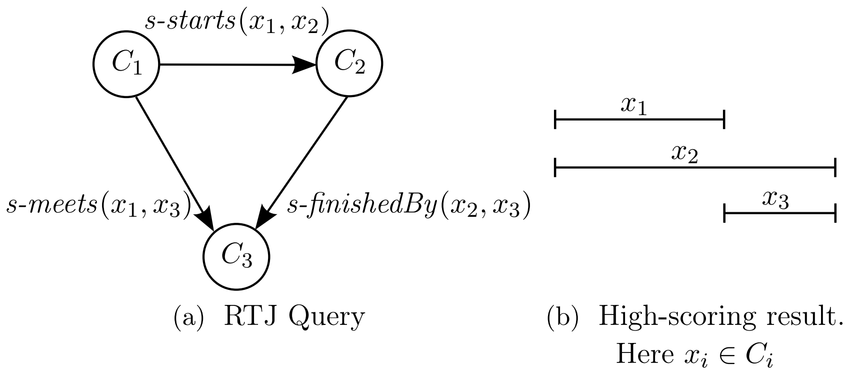

$s\text{-}starts(x_1, x_2)$
$C_1$
$C_2$
$C_3$
$s\text{-}meets(x_1, x_3)$
$s\text{-}finishedBy(x_2, x_3)$
$x_1$
$x_2$
$x_3$
(a) RTJ Query
(b) High-scoring result. Here $x_i \in C_i$

Figure 2.5: Example of RTJ query

query edge $(i, j) \in E$. $\mathcal{S}$ could be any monotone function such as the weighted sum as it is commonly the case in ranked aggregation [46, 51, 72, 95].

For example, we can express a 3-way query that returns a tuple $(x, y, z)$ where $x \in C_1$, $y \in C_2$ and $z \in C_3$ and the score of $(x, y, z)$ is computed as an aggregation of its partial scores for query predicates *s-starts*$(x, y)$ and *s-meets*$(y, z)$.

Although we use the term "join" to refer to our queries, their expressiveness goes beyond traditional relational joins. Our queries are not compositional in the sense of a relational join since their results are not intervals but tuples of any length (corresponding to the number of vertices in the query). Our queries can express any combination of interval collections with any scored predicates including chain queries and queries containing cycles.

Figure 2.5a illustrates a RTJ query featuring three predicates, three collections and a cycle. This query returns triplets of intervals. If scores are aggregated using the average, the high-scoring results have the form illustrated in Figure 2.5b.

#### 2.3.1.2 Ranked Temporal Join (RTJ) problem

Given an n-ary temporal join query $Q = (V, E)$ expressed over a set of collections $C_1, \ldots, C_m$ corresponding to query vertices in $V$ and temporal predicates $s\text{-}p_{(i,j)}()$ associated to each edge $(i, j) \in E$, our problem is to find a top-$k$ set of tuples of the form $(x_1, \ldots, x_n)$, $x_i \in C_i$, ranked by (descending) order of $\mathcal{S}_{(i,j)\in E}(s\text{-}p_{(i,j)}(x_i, x_j))$.

### 2.3.2 Our Approach for Processing Batch Temporal Joins

We present *TKIJ*, our approach for evaluating Top-K Interval Joins, that efficiently finds the set of $k$ *best* results for an RTJ query $Q$. We first provide an overview of *TKIJ*, then we give each step in detail.

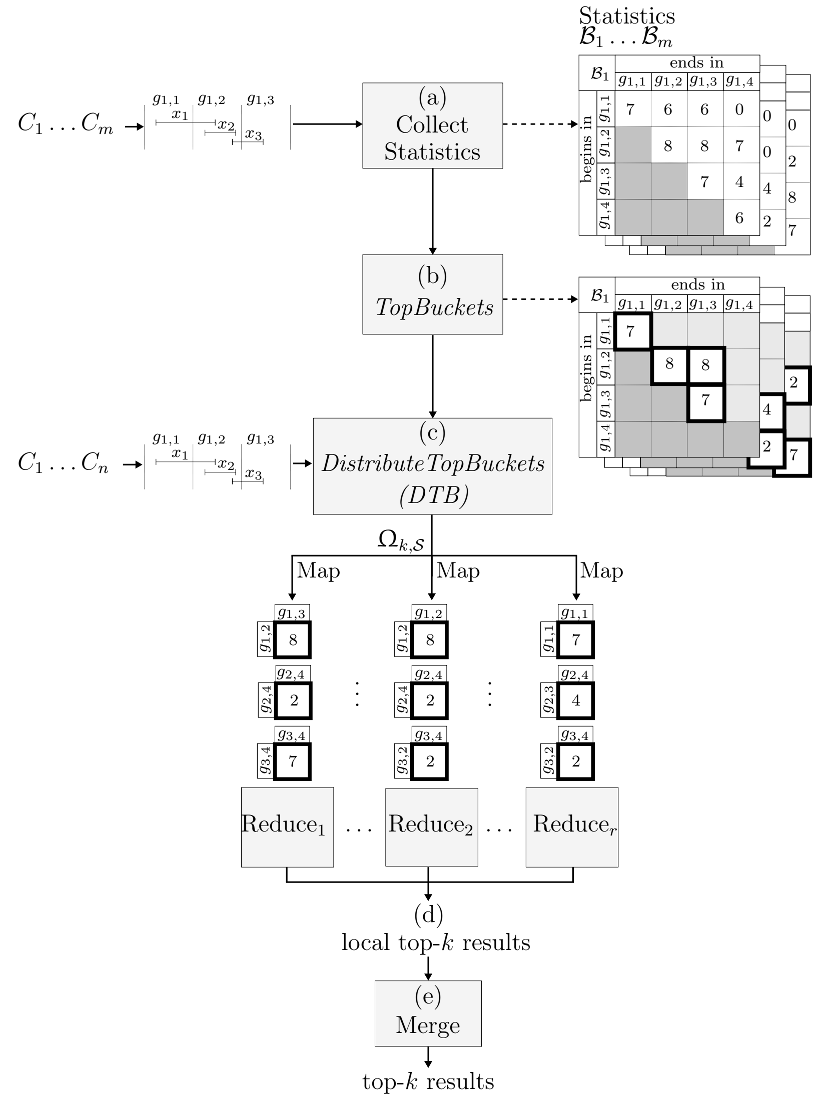


Figure 2.6: Overview of *TKIJ*

### 2.3.2.1 Overview of TKIJ

Figure 2.6 summarizes *TKIJ*. Given a set of interval collections $C_1 \dots C_m$, *TKIJ* executes a query-independent pre-processing phase to collect statistics on intervals' distribution. This phase partitions time into granules and computes *buckets* for each collection (a). A bucket associated to a collection $C_i$ corresponds to a pair of granules, and contains the number of intervals of $C_i$ starting at one granule and ending at another. Given a query $Q$, these statistics are used to evaluate *bucket combinations* that should be processed in order to obtain top-$k$ results (b). *TKIJ* relies on a constraint programming solver to compute score bounds for each bucket combination and uses those bounds to prune combinations that do not contain top-$k$ results. The third phase is the actual join processing which relies on two Map-Reduce jobs. The first job assigns a subset of buckets to each reducer $r_j$ (c) which then processes locally the RTJ query, returning local top-$k$ results (d). This assignment aims at reducing data replication to limit I/O, and leverages score bounds to distribute high-scoring results evenly so that each reducer can quickly prune low-ranking results. The second Map-Reduce job merges all local results into a single query output (e).

#### 2.3.2.2 Statistics collection

*TKIJ* pre-processes each dataset once in order to collect statistics which are then used to optimize the execution of any RTJ query on this dataset. These statistics maintain a matrix $\mathcal{B}_i$ representing the distribution of endpoints of intervals in each collection $C_i$. *TKIJ* partitions the time range of each $C_i$ into a set of contiguous *granules*. We adopt a uniform partitioning which has been shown to be appropriate for temporal joins [29, 35, 54].

As illustrated in Figure 2.6a, each matrix entry records the cardinality of a bucket, where a bucket $b_{i,l,l'} = (g_{i,l}, g_{i,l'})$ contains all intervals of $C_i$ that start in $g_{i,l}$ and end in $g_{i,l'}$: $\mathcal{B}_i[l][l'] = |b_{i,l,l'}| = |\{x \in C_i, \underline{x} \in g_{i,l} \wedge \overline{x} \in g_{i,l'}\}|$. As an example, given $g_{1,1} = [10, 20]$ and $g_{1,2} = [20, 30]$, the matrix entry for $b_{1,1,2} = (g_{1,1}, g_{1,2})$ indicates 6 intervals starting in $[10, 20]$ and ending in $[20, 30]$.

Range partitioning is a common approach in temporal join processing [29, 35, 54, 88]. The rationale is that intervals having similar endpoints are likely to satisfy similar join predicates. For example, most previous studies, that focus on *intersection* joins, leverage partitions to avoid pairs of intervals that are guaranteed not to intersect. Similarly, *TKIJ* relies on these statistics to obtain information on the distribution of intervals within buckets and prune the search space of *any* RTJ query.

Statistics are computed in a single Map-Reduce phase. Each mapper reads a fraction of the data and maintains a local matrix per collection. Matrices are then aggregated in the reduce phase, and the reducer responsible for collection $C_i$ outputs a final matrix $\mathcal{B}_i$. While we focus in this thesis on the case of statistics computed from scratch for a new dataset, we can easily handle updates by applying the same process on the inserted/deleted data.

#### 2.3.2.3 Selection of bucket combinations

We now describe how *TKIJ* uses pre-computed statistics to estimate score bounds on candidate results. Then, we present how score bounds are used to avoid computing unnecessary results while we guarantee to return the exact top-$k$ results. We further develop several processing strategies that aim to tackle computational costs raised by this pruning step.

**Estimating score bounds.** Processing an RTJ query $Q$ requires to return the top-$k$ tuples $(x_1, \ldots, x_n)$, $x_i \in C_i$ according to a scoring function $\mathcal{S}$. Since any tuple $(x_1, \ldots, x_n)$ is a potential answer, we investigate how to reduce the amount of data processed using scores.

We use $\omega = (b_{1,l_1,l'_1}, \ldots, b_{n,l_n,l'_n})$ to denote a bucket combination, $\omega.nbRes = \prod_{i=1}^{n} |b_{i,l_i,l'_i}|$ the total number of results that can be obtained from a bucket combination $\omega$, and $\Omega$ the set of all combinations. We define the score upper and lower bounds in each $\omega$, denoted $\omega.UB$ and $\omega.LB$ as follows:

*Score Bounds.* The score upper-bound (resp. lower-bound) $\omega.UB$ (resp. $\omega.LB$) of a bucket combination $\omega = (b_{1,l_1,l'_1}, \ldots, b_{n,l_n,l'_n})$ is the upper-bound (resp. lower-bound) of $\mathcal{S}_{(i,j)\in E}(s\text{-}p_{(i,j)}(x_i, x_j))$ where $\underline{x_i} \in g_{i,l_i}, \overline{x_i} \in g_{i,l'_i}, \forall i \in 1 \ldots n$.

As an example, suppose that query $Q$ features a predicate $s\text{-}meets_{(1,2)}(x, y)$ where $x \in C_1$ and $y \in C_2$, using scoring parameters $(\lambda_{equals}, \rho_{equals}) = (4, 8)$. Collected statistics show 6 intervals in bucket $b_{1,1,2} = ([10, 20], [20, 30])$ for $C_1$ and 7 intervals in bucket $b_{2,2,3} = ([20, 30], [30, 40])$ for $C_2$. We build the bucket combination $\omega = (b_{1,1,2}, b_{2,2,3})$. Then, we can derive bounds on the score $\mathcal{S}(x, y) = s\text{-}meets(x, y)$ of a result $(x, y) \in \omega$. The maximum possible score is 1 (e.g. with $(x, y) = ([12, 25], [25, 35])$), and the minimum score is 0.25 (with $(x, y) = ([15, 20], [30, 35])$). Hence, $\omega.UB = 1, \omega.LB = 0.25$. Thus, 42 results in $\omega$ have a score in $[0.25, 1]$.

*TKIJ* relies on a constraint programming solver as a generic approach to compute score bounds for any combination of predicates. Computing score bounds for a bucket combination requires to solve the following problem:

*Bounds Problem.* Let $\omega = (b_{1,l_1,l'_1}, \ldots, b_{n,l_n,l'_n})$. Find $(x_1, \ldots, x_n)$ s.t.:

$$\begin{aligned} \textit{max (resp. min)} \quad & \mathcal{S}_{(i,j)\in E}(s\text{-}p_{(i,j)}(x_i, x_j)) \\ \textit{s.t.} \quad & \underline{x}_i \in g_{i,l_i} \ \forall i \in 1 \ldots n && (2.1) \\ & \overline{x}_i \in g_{i,l'_i} \ \forall i \in 1 \ldots n && (2.2) \end{aligned}$$

Each $x_i \in C_i$ is mapped to a decision variable $\mathtt{x}_i$. For each partial score $s\text{-}p_{(i,j)}(x_i, x_j)$, we create an intermediate variable $\mathtt{s\text{-}p_{ij}}$. For all $(i, j) \in E$, we impose that the variables $\mathtt{x}_i, \mathtt{x}_j$ and $\mathtt{s\text{-}p_{ij}}$ satisfy the constraints $\{\mathtt{C}_{ij} : \mathtt{s\text{-}p_{ij}} = s\text{-}p_{(i,j)}(\mathtt{x}_i, \mathtt{x}_j)\}$. Then, we create a variable $\mathtt{score}$ and impose $\mathtt{C}_s : \mathtt{score} = \mathcal{S}_{(i,j)\in E}(\mathtt{s\text{-}p_{ij}})$. The solver then maximizes (and minimizes in the case of a lower-bound) $\mathtt{score}$ such that constraint $\mathtt{C}_s$, all constraints $\mathtt{C}_i$ and all constraints on decision variables (2.1-2.2) are satisfied. While we virtually allow any temporal predicates, in practice, predicate implementation depends on the range of constraints supported by the server used in the implementation.

**Pruning bucket combinations.** *TKIJ* leverages computed score bounds to reduce computation cost by eliminating results that are guaranteed not to be in the top-$k$. To do so, it computes $\Omega_{k,\mathcal{S}} \subseteq \Omega$, a subset of the search space that is sufficient to guarantee correctness. We define $\Omega_{k,\mathcal{S}}$ as follows:

*Top Buckets.* The set of Top Buckets $\Omega_{k,\mathcal{S}}$ is a subset of $\Omega$ satisfying the following conditions:

- $\forall \omega \in \Omega \setminus \Omega_{k,\mathcal{S}} \; \exists \Psi \subseteq \Omega_{k,\mathcal{S}}$:

- $\forall \omega' \in \Psi \; \omega'.LB \geq \omega.UB$
- $\sum_{\omega' \in \Psi} \omega'.nbRes \geq k$

This definition ensures that whenever a bucket combination $\omega$ is pruned, there are at least $k$ results from $\Omega_{k,\mathcal{S}}$ with a score higher than results generated from $\omega$. Note that $\Omega_{k,\mathcal{S}}$ is not unique: given a valid set of bucket combinations, any super-set is also valid. To compute $\Omega_{k,\mathcal{S}}$, we design the *getTopBuckets* algorithm (Algorithm 1). Algorithm *getTopBuckets* uses as input a set of bucket combinations whose score bounds are pre-computed. It first computes a lower bound *kthResLB* on the score of the $k^{th}$ result (Lines 1-6). Then, it keeps only bucket combinations whose score upper-bound is greater than *kthResLB* (Lines 7-13). The process safely stops in Line 11 since no bucket combination outside the collected set has results with score above *kthResLB*.

**Algorithm 1** *getTopBuckets*

```
Input: $k$, $\Omega$ list of bucket combinations with UB and LB
Output: $\Omega_{k,\mathcal{S}}$
 1: Sort $\Omega$ by descending $LB$
 2: $collectedResults = 0$
 3: for $\omega \in \Omega$
 4:     $collectedResults \mathrel{+}= \omega.nbRes$
 5:     $kthResLB = \omega.LB$
 6:     if $collectedResults \geq k$ then break
 7: Sort $\Omega$ by descending $UB$
 8: $\Omega_{k,\mathcal{S}} \leftarrow \emptyset$, $collectedResults = 0$
 9: for $\omega \in \Omega$
10:     if $collectedResults \geq k$ and $\omega.UB \leq kthResLB$
11:         break
12:     $\Omega_{k,\mathcal{S}} \leftarrow \Omega_{k,\mathcal{S}} \cup \omega$
13:     $collectedResults \mathrel{+}= \omega.nbRes$
    return $\Omega_{k,\mathcal{S}}$
```

Pruning unnecessary results is a two-step process, coined *TopBuckets*. A first step computes score bounds for bucket combinations using a solver. Then, a second step executes *getTopBuckets* that uses those bounds to eliminate unnecessary results. In our setting, all $(x_1, \ldots, x_n)$ combinations are potential answers, and we cannot employ traditional top-$k$ techniques to prune the search space. *TopBuckets* addresses this challenge using pre-computed statistics to locate high-scoring answers. Still, a new challenge is to limit the overhead of *TopBuckets* due to computing score bounds for bucket combinations. In the following section, we discuss this challenge and possible solutions.

**Algorithm 2** Strategies LOOSE, TWO-PHASE

```
Input: Boolean onePhase, Buckets $\{b_{i,l_i,l'_i} : i \in 1 \dots n\}$
Output: $\Omega_{k,\mathcal{S}}$
1: $L_2 \leftarrow$ all bucket pairs $(b_i, b_j)$ s.t. $(i,j) \in E$
2: for all $\omega$ in $L_2$
3:     Compute $\omega.UB$, $\omega.LB$ using solver
4: for $\omega$ in $\Omega$
5:     Compute $\omega.UB$, $\omega.LB$ using score bounds from $(b_i, b_j) \in L_2$
6: $L_m \leftarrow$TopBuckets($\Omega$)
7: if onePhase then return $L_m$
8: for $\omega \in L_m$
9:     Compute $\omega.UB$, $\omega.LB$ using solver
   return TopBuckets($L_m$)
```

**TopBuckets Strategies.** A first straightforward approach to find $\Omega_{k,\mathcal{S}}$ is to build all possible bucket combinations $\Omega$, compute their score bounds using a solver and then use *getTopBuckets* to prune useless ones. In this strategy, coined BRUTE-FORCE, a large number of $n$-tuples of buckets are assigned a score by the solver (with $g$ granules per collection, $|\Omega|$ is $\mathcal{O}(g^{2n})$). Moreover, for each combination, $2n$ decision variables need to be assigned by the solver. Thus, as $n$ or $g$ increase, BRUTE-FORCE becomes inefficient.
To tackle that, we propose the LOOSE strategy (Algorithm 2 with the flag onePhase set true.) LOOSE first builds all bucket pairs $(b_{i,l_i,l'_i}, b_{j,l_j,l'_j})$ for each scored predicate $(i,j) \in E$ (Line 1). Score bounds are then computed by the solver (Line 3) for each pair. Then, LOOSE builds $n$-tuples of buckets (Lines 4-5). For each $\omega = (b_{1,l_1,l'_1}, \dots, b_{n,l_n,l'_n})$, we obtain score bounds using bounds computed for each pair $(b_{i,l_i,l'_i}, b_{j,l_j,l'_j})$. To calculate correct bounds, we rely on the monotonicity of $\mathcal{S}$. Without loss of generality, suppose that $\mathcal{S}$ is monotonically increasing. In the expression of $\mathcal{S}$, we replace each partial score $s\text{-}p_{(i,j)}(x_i, x_j)$ with the upper bound of the corresponding pair of bucket. Therefore $\mathcal{S}_{(i,j)\in E}((b_{i,l_i,l'_i}, b_{j,l_j,l'_j}).UB)$ is a correct upper-bound for $\omega$. Then, LOOSE runs *getTopBuckets* on the generated bucket combinations (Line 6) and returns the selected combinations. The rationale behind LOOSE is that processing time can decrease significantly because (i) fewer bucket combinations need to be assigned a score bound (their number is $\mathcal{O}(|E| \cdot g^4)$) and (ii) the solver needs to assign only 4 variables when computing score bounds. A drawback of LOOSE is that the aggregation of bounds using $\mathcal{S}$ may result in loose bounds. We illustrate that in the following example.

*Example* 1. Figure 2.7 depicts a dataset with three buckets $b_1, b_2, b_3$ from $C_1, C_2, C_3$. We have $b_1 = (g_1, g_2)$, $b_2 = (g_2, g_3)$, $b_3 = (g_3, g_3)$, where $g_1 = [10, 20], g_2 = [20, 30], g_3 = [30, 40]$. Our query features scored predicates $s\text{-}starts_{(1,2)}(x, y)$ and $s\text{-}starts_{(2,3)}(y, z)$ and our aggregation function is the normalized sum. We use the scoring parameters $\{(\lambda_{equals}, \rho_{equals}),$

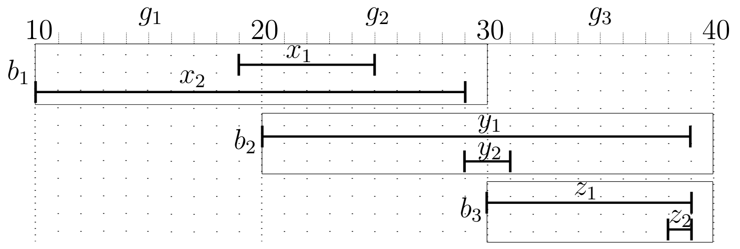


Figure 2.7: Example of Bucket Combinations

$(\lambda_{greater}, \rho_{greater}) = \{(1,3),(0,4)\}$. LOOSE first computes bounds for $\omega_1 = (b_1, b_2)$. We have $\omega_1.UB = 1$ (because $s\text{-}starts_{(1,2)}(x_1, y_1) = 1$), and $\omega_1.LB = 0$ ($s\text{-}starts_{(1,2)}(x_1, y_2) = 0$). Then, LOOSE computes bounds for $\omega_1 = (b_2, b_3)$. We have $\omega_2.UB = 1$ ($s\text{-}starts_{(2,3)}(y_2, z_1) = 1$), and $\omega_2.LB = 0$ ($s\text{-}starts_{(2,3)}(y_2, z_2) = 0$). Then, LOOSE merges combinations $\omega_1, \omega_2$ in $\omega_3 = (b_1, b_2, b_3)$, and computes $\omega_3.UB = \mathcal{S}(1,1) = 1$, $\omega_3.LB = \mathcal{S}(0,0) = 0$. Yet, BRUTE-FORCE, computes $\omega_3.UB = 0.5$ because there is no $(x, y, z)$ such that $s\text{-}starts_{(1,2)}(x, y) = s\text{-}starts_{(2,3)}(y, z) = 1$ given the buckets' bounds: it is impossible to have both $equals(\underline{x}, \underline{y}) = 1$ and $equals(\underline{y}, \underline{z}) = 1$ with $\underline{x} \in g_1, \underline{y} \in g_2$ and $\underline{z} \in g_3$. Thus, in this example, LOOSE returns an exact lower-bound and a loose upper-bound, while BRUTE-FORCE returns tight bounds.

These observations lead to propose a third strategy TWO-PHASE, that combines BRUTE-FORCE and LOOSE. TWO-PHASE is executed by Algorithm 2 when the flag onePhase is set to false. First, TWO-PHASE computes loose bounds to eliminate some bucket combinations (Lines 1-7), identically to LOOSE. Then, TWO-PHASE refines the bounds of the remaining combinations (Lines 8-9) to obtain exact bounds. The rationale behind TWO-PHASE is that its first phase may help reduce the number of bucket combinations that need to be assigned a score in the second phase, thus improving the solver's running time. Unlike LOOSE, TWO-PHASE returns tight bounds thanks to the second phase of the solver.

When selecting bucket combinations, *TKIJ* runs *TopBuckets* using one of the three strategies presented. Each strategy (i) employs the solver and (ii) executes *getTopBuckets* once or twice using loose or tight score bounds on bucket combinations.

#### 2.3.2.4 Distributed Top-k Join Processing

The *TopBuckets* process generates $\Omega_{k,\mathcal{S}}$, a set of bucket combinations that are sufficient to accurately compute the top-$k$ results. We now describe how *TKIJ* computes the top-$k$ results (Steps (c)-(d)-(e) in Figure 2.6). We implement *TKIJ* on Map-Reduce [34]. Given a set of $r$ reducers, *TKIJ* assigns each bucket combination $\omega \in \Omega_{k,\mathcal{S}}$ to a single reducer $r_j, j \in 1 \ldots r$, that processes results in $\omega$. The main challenge in distributed join processing is to devise an efficient workload assignment function. When performing large-scale joins, I/O often constitutes a major bottleneck. We first review existing assignment algorithms, then we consider the specifics of distributed top-$k$ computation and show that it is essential to take scores into account when dividing the workload. We present *DistributeTopBuck-*

**Algorithm 3** DistributeTopBuckets (DTB)

```
Input: $\Omega_{k,\mathcal{S}}$
Output: $M$: assignments (bucket, reducer)
1: Sort $\Omega_{k,\mathcal{S}}$ by descending order of $\omega.UB$
2: $avgRes = \frac{\sum_{\omega \in \Omega_{k,\mathcal{S}}} \omega.nbRes}{r}$
3: for all $\omega \in \Omega_{k,\mathcal{S}}$
4:     $r_j = getReducer(avgRes, \omega)$
5:     for all bucket $b \in \omega$                    ▷ Assign buckets in $\omega$ to $r_j$
6:         $M \leftarrow M \cup (b, r_j)$
   return $M$
```

*ets*, a novel function that focuses on assigning high-scoring results to each reducer, while minimizing I/O cost as a secondary objective. Finally, we present how an RTJ query is processed using appropriate Map-Reduce algorithms.

**Existing I/O optimizations.** When different reducers require the same chunk of data, this data is replicated in the shuffle phase of Map-Reduce, which increases input cost. Several distributed join algorithms, such as *RCCIS* [29] and the work of Afrati et al. [4] specifically aim at reducing that cost. In *TKIJ*, this corresponds to different reducers being assigned bucket combinations involving the same bucket. Other approaches focus on assigning a balanced load to each reducer [29, 97]. This ensures that the number of results generated by each reducer is comparable, so that no reducer will have a larger workload in output dominated tasks. Finally, some algorithms optimize both input and output costs simultaneously [134]. All these approaches are not directly applicable to our settings. They achieve optimizations for specific queries (equi-join [4], 2-way $\theta$-join [97], m-way $\theta$-join [134]). One close related work to ours [29] reduces I/O cost by leveraging the Boolean interpretation of Allen predicates. That is not directly applicable to scored predicates.

**Top-k optimizations.** *TKIJ* significantly differs from standard Map-Reduce-based join processes due to its ranked semantics. In *TKIJ*, each reducer processes a full RTJ query locally using the bucket combinations it receives (Figure 2.6d). Hence, it is important to ensure that each reducer quickly identifies high-scoring results as it is usually the case in top-$k$ processing [45, 46, 51, 110]. Therefore, the assignment of bucket combinations to reducers favors an even distribution of high-scoring results.

**DTB algorithm.** *TKIJ* relies on the *DistributeTopBuckets* algorithm (Algorithm 3) to assign bucket combinations from $\Omega_{k,\mathcal{S}}$ to reducers. Following the principles described above, *DTB* increases the probability that each reducer receives a fair share of high-scoring results. This step relies on the knowledge, for each bucket combination, of the number of

**Algorithm 4** getReducer

```
Input: $\omega$ bucket combination to assign, $avgRes$ average number of results per reducer
Output: reducer to which $\omega$ has to be assigned
 1: $min\_\omega\_assigned = +\infty$
 2: for $j = 1$ to $r$          ▷ Retrieve the minimum amount of bucket combinations assigned
 3:     if $r_j.nbRes < 2 \times avgRes$
 4:         $min\_\omega\_assigned = \min\{|\Omega_{r_j}|, min\_\omega\_assigned\}$
 5: $minCost = +\infty$
 6: for $j = 1$ to $r$          ▷ Find the best reducer wrt $inCost(\omega, r_j)$
 7:     if $r_j.nbRes < 2 \times avgRes \wedge |\Omega_{r_j}| = min\_\omega\_assigned$
 8:         if $inCost(r_j, \omega) < minCost$
 9:             bestReducer=$r_j$
10:             $minCost = inCost(r_j, \omega)$
    return bestReducer
```

results generated, as well as their score bounds. *DTB* first sorts $\Omega_{k,\mathcal{S}}$ by descending order of score upper-bound (Line 1) to access them according to their likelihood of generating high-scoring results. It then assigns each bucket combination to a reducer using the *getReducer* function (Algorithm 4), which returns a reducer among the ones that were assigned the fewest bucket combinations so far.

Furthermore, *DTB* opportunistically optimizes I/O cost. First, in the worst case, a reducer evaluates all the results it is assigned. If $\Omega_{r_j}$ is the set of bucket combinations assigned to a reducer $r_j$, then $\sum_{\omega \in |\Omega_{r_j}|} \omega.nbRes$ would be computed. *DTB* first computes the average number of results assigned to reducers (Algorithm 3, Line 2). Then *getReducer* ensures that reducers that are already assigned more than twice the average number of results are discarded (Algorithm 4, Lines 3, 7). This heuristic limits imbalance in a worst-case scenario. In the case, where several reducers have received the same number of bucket combinations, *getReducer* selects the reducer that was already assigned the largest fraction of current $\omega$ from previous overlapping bucket combinations (Algorithm 4, Lines 8-10). For a given $w$, *getReducer* evaluates for each reducer $r_j$ the input cost of assigning $w$ to $r_j$ using $inCost(r_j, \omega)$. We define $inCost(r_j, \omega) = \sum_{b_{i,l,l'} \in \omega} |b_{i,l,l'}| \cdot \Phi(r_j, b_{i,l,l'})$ where $\Phi(r_j, b_{i,l,l'})$ returns 1 if $b_{i,l,l'}$ was already assigned to $r_j$, else 0. This process favors assignments that reduce replication cost. Having selected the reducer $r_j$ that has to be assigned the current bucket combination $\omega$, *DTB* stores all the assignments $(b_{i,l,l'}, r_j)$ where $b_{i,l,l'} \in \omega$ (Algorithm 3, Lines 5-6). These assignments determine to which reducers buckets are communicated, ensuring both output correctness and join processing efficiency.

**Join Processing.** The final phase of *TKIJ* first runs *DTB* using $\Omega_{k,\mathcal{S}}$ to determine data distribution among reducers. Then, a Map-Reduce phase processes the RTJ query locally. For each input interval $x$, a mapper computes the bucket $b_{i_x,l_x,l'_x}$ in which $x$ falls. Then, $x$

| **Id** | **Scored Temporal Predicates In $\mathcal{Q}$.** $x_i \in C_i \ \forall i \in 1 \ldots n.$ |
|---|---|
| $\mathcal{Q}_{b,b}$ | *s-before*$(x_1, x_2)$, *s-before*$(x_2, x_3)$ |
| $\mathcal{Q}_{f,f}$ | *s-finishedBy*$(x_1, x_2)$, *s-finishedBy*$(x_2, x_3)$ |
| $\mathcal{Q}_{o,o}$ | *s-overlaps*$(x_1, x_2)$, *s-overlaps*$(x_2, x_3)$ |
| $\mathcal{Q}_{s,f,m}$ | *s-starts*$(x_1, x_2)$, *s-finishedBy*$(x_2, x_3)$, *s-meets*$(x_1, x_3)$ |
| $\mathcal{Q}_{s,s}$ | *s-starts*$(x_1, x_2)$, *s-starts*$(x_2, x_3)$ |
| $\mathcal{Q}_{b*}$ | *s-before*$(x_1, x_2), \ldots,$ *s-before*$(x_1, x_n)$ |
| $\mathcal{Q}_{o*}$ | *s-overlaps*$(x_1, x_2), \ldots,$ *s-overlaps*$(x_1, x_n)$ |
| $\mathcal{Q}_{m*}$ | *s-meets*$(x_1, x_2), \ldots,$ *s-meets*$(x_1, x_n)$ |
| $\mathcal{Q}_{f,b}$ | *s-finishedBy*$(x_1, x_2)$, *s-before*$(x_2, x_3)$ |
| $\mathcal{Q}_{o,m}$ | *s-overlaps*$(x_1, x_2)$, *s-meets*$(x_2, x_3)$ |
| $\mathcal{Q}_{s,m}$ | *s-starts*$(x_1, x_2)$, *s-meets*$(x_2, x_3)$ |
| $\mathcal{Q}_{jB,jB}$ | *s-justBefore*$(x_1, x_2)$, *s-justBefore*$(x_2, x_3)$ |
| $\mathcal{Q}_{sM,sM}$ | *s-shiftMeets*$(x_1, x_2)$, *s-shiftMeets*$(x_2, x_3)$ |

Table 2.1: Experiments: RTJ queries

is communicated to all reducers $r_j$ that received $b_{i_x,l_x,l'_x}$. That way, each reducer $r_j$ receives its share of input data, and a list of bucket combinations $\Omega_{r_j} \subseteq \Omega_{k,\mathcal{S}}$ whose results are potential top-$k$ candidates. Once each reducer has processed locally the RTJ query, we run an additional Map-Reduce phase (Step (e) in Figure 2.6), that merges local results and returns the final top-$k$ answers.

## 2.3.3 Experiments

### 2.3.3.1 Settings

**Platform.** We conduct experiments on an 8-node industrial cluster with 6 workers. Each worker has 1 Intel Xeon E5-2650L (8 cores), 32GB RAM, 5TB disk. Machines run Centos 6.6 with Cloudera 5.2.5 and Hadoop 2.5.0. All presented results are averages of 5 consecutive runs.

**Statistics collection.** We use the same number of granules $g$ for each collection. We observed that only the number of interval $|C_i|$ per collection had a significant impact on statistics collection time. Statistics collection lasted between 28s for $|C_i| = 2\times10^5$ and 36s for $|C_i| = 5\times10^6$. Since this task is only executed once as a pre-processing for a given dataset, we do not include it in query evaluation time.

| Id | $(\lambda_{equals}, \rho_{equals})$ | $(\lambda_{greater}, \rho_{greater})$ |
|---|---|---|
| $\mathcal{P}_1$ | (4,16) | (0,10) |
| $\mathcal{P}_2$ | (0,16) | (2,8) |
| $\mathcal{P}_3$ | (4,12) | (0,8) |
| $\mathcal{P}_B$ | (0,0) | (0,0) |

Table 2.2: Experiments: scored predicates parameters

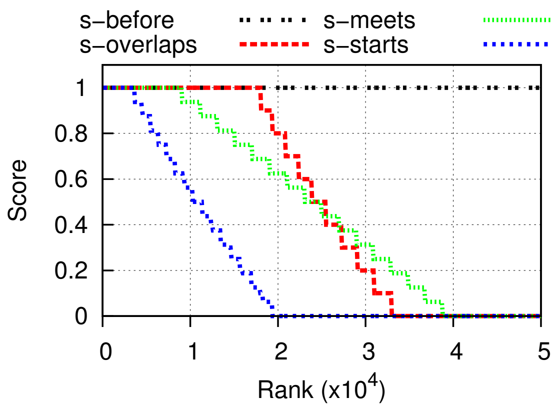
**Selection of bucket combinations.** We implement a distributed and multi-threaded version of *TopBuckets*. We split the set of buckets $\mathcal{B}_1$ into 6 equal-sized groups $\mathcal{B}_{1,j}, j \in 1 \ldots 6$. Then each worker $j \in 1 \ldots 6$ runs a local version of *TopBuckets* using buckets in $\mathcal{B}_{1,j}$ and all buckets in $\mathcal{B}_i, i \in 2 \ldots n$. Thus, each worker has as input a disjoint set of possible bucket combinations. We use Choco [104], a software for constraint programming to compute score bounds for each bucket combination. Each execution of the solver is handled in a separate thread. A second phase of *TopBuckets* merges local *TopBuckets* results on a selected worker and returns the set of best bucket combinations $\Omega_{k,S}$.

**Distributed join processing.** We run *TKIJ* using 24 reducers. Local query execution accesses the received bucket combinations by descending order of score upper-bounds. It then uses R-Trees to access intervals in memory. For an interval $x_i$ and a score value $v$, it queries the R-Tree and returns only intervals $x_j$ s.t. $s\text{-}p_{(i,j)}(x_i, x_j) \geq v$.

**Queries.** Tables 2.1 and 2.2 summarize queries and score parameters. We use $\mathcal{S} = \frac{\sum_{(i,j) \in E} s\text{-}p_{(i,j)}(x_i,x_j)}{|E|}$ to compute the score of a query result $(x_1, \ldots, x_n)$. Unless otherwise specified, $k = 100$.

### 2.3.3.2 Summary of Results

We show that *TKIJ* processes various RTJ queries efficiently on both synthetic data and real network traffic logs. *TKIJ* scales to collections of up to 5 million intervals ($|C_i| \in [1M, 5M]$) and efficiently returns the top-$k$ results for $k \in [10, 10^5]$. We show that our workload distribution approach, *DistributeTopBuckets*, that fairly distributes high-scoring results, outperforms a more naive approach based on the *LPT* algorithm. That is particularly useful for queries that return few high-scoring results, such as those featuring equality-based predicates (e.g *starts*). We observe that the efficiency of *DistributeTopBuckets* decreases with coarser statistics. We also show that as the number of collections or the number of predicates in a query vary, *TopBuckets* efficiently prunes buckets combinations. Experiments on network traffic data show that on queries featuring *before* or *overlaps*, *TopBuckets* can select few bucket combinations guaranteeing *TKIJ* to return high-scoring results. We observe that a higher number of granules $g$ (finer statistics) helps prune unnecessary results,



Parameters: $|C_i| = 10^4, \mathcal{P} = \mathcal{P}_1$

Figure 2.8: Synthetic data: score distribution

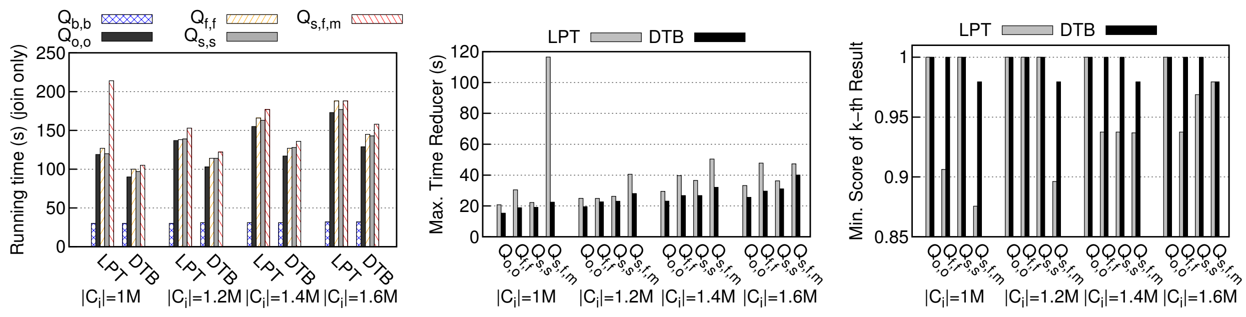
which improves overall join processing time. However, it also makes pruning computation with *TopBuckets* slower. Hence, we run experiments to find a sweet-spot value for $g$.

#### 2.3.3.3 Synthetic Data

To generate synthetic data, we use the same parameters as in previous work [29]. We use a pseudo-random uniform generator to get intervals' startpoints and lengths in specified ranges (respectively $s = [0, 10^5]$ and $w = [1, 100]$). Intervals' endpoints are integers. We vary the number $|C_i|$ of intervals per collection, and the number $n$ of collections. A collection of 5M intervals measures ≈113MB (text format).

**Score Distribution.** We conduct preliminary experiments to get an insight of the score distribution of a set of results using scored temporal predicates. Especially, we want to measure the share of results that have a high score. We want to verify that the fewer the number of high-scoring results, the less likely a worker will receive high scoring results from $DTB$. We compute all the combinations $(x_1, x_2) \in C_1 \times C_2$ and we evaluate each result with scored predicates *s-before*$(x_1, x_2)$, *s-meets*$(x_1, x_2)$, *s-overlaps*$(x_1, x_2)$ and *s-starts*$(x_1, x_2)$.

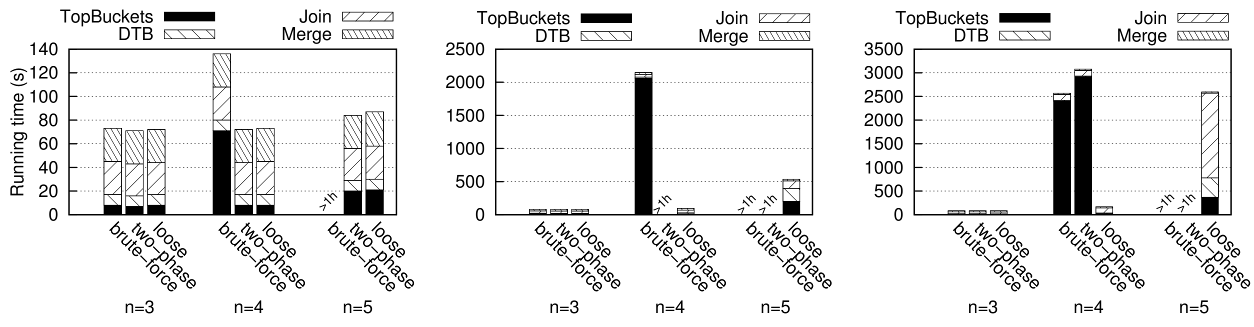
On Figure 2.8, we plot scores for the top-50000 results. *s-before* is the predicate with the largest number of high-scoring results (scores of top-50000 results equal 1.0), since a single inequality on endpoints is required. More high-scoring results can be found with *s-overlaps* (≈18,000 results), that evaluates only inequality on endpoints, than with *s-meets* (≈9000), where an equality is required. Fewer results are assigned a high score when *s-starts* is used since it requires both equality and inequality on endpoints. Thus, we can expect a faster join processing on queries using only inequality-based predicates where more high-scoring results can be found, since high-scoring results favor early termination of local top-$k$ processing.



(a) Running Time (b) Max Running Time of Reducers (c) Minimum Score of k-th Result

Parameters: $g = 20, k = 1000, \mathcal{P} = \mathcal{P}_2$, *TopBuckets* : LOOSE

Figure 2.9: Experiments on synthetic data: workload distribution



(a) $\mathcal{Q}_{b^*}$ (b) $\mathcal{Q}_{o^*}$ (c) $\mathcal{Q}_{m^*}$

Parameters: $g = 15, k = 100, |C_i| = 2 \times 10^5, \mathcal{P} = \mathcal{P}_1$

Figure 2.10: Experiments on synthetic data: detailed execution time, all *TopBuckets* strategies

**Workload Distribution.** We conduct experiments to validate our workload distribution approach. We analyze executions of *TKIJ* using *DTB*, our workload distribution algorithm, and using a more straightforward algorithm.

**LPT.** In the context of task scheduling, the *LPT* (*Longest Processing Time*) heuristic aims to minimize scheduling time on parallel machines [39]. *LPT* executes tasks in descending order of processing time. In our context, a naive approach would minimize the maximum number of candidate join results that a reducer has to process, so as to reduce the duration of the longest task. With *LPT*, bucket combinations are analogous to tasks that we want to assign to a set of reducers. We sort the set of bucket combinations by descending order of number of results ($\omega.nbRes$) and assign each one to the least loaded reducer.

**Results.** Figure 2.9a presents the running time of the join phase on all queries, where $|C_i|$ varies from $1 \times 10^6$ to $1.6 \times 10^6$. On $\mathcal{Q}_{b,b}$, running time is identical for $LPT$ and $DTB$, since a single bucket combination is selected and a large number of results with maximum score can be quickly found during the join phase. On other queries, $DTB$ outperforms $LPT$ for two reasons. Firstly, $LPT$ incurs a higher shuffle cost (on average 43% higher). When assigning a bucket combination to a reducer, $DTB$ favors assignments that lessen shuffle cost. $LPT$ favors the assignment of bucket combinations with a large number of results to the least loaded reducers. Hence, buckets have a higher probability to be sent to several reducers with $LPT$ than with $DTB$. Secondly, $LPT$ does not necessarily give a fair share of high-scoring results to each reducer. Figure 2.9b shows the running time of the longest reducer task (we omit $\mathcal{Q}_{b,b}$ where $LPT$ and $DTB$ perform equally for the reason exposed above). $DTB$ always outperforms $LPT$ because it increases the probability that all reduce tasks terminate early since they can all find high-scoring results. This difference is exacerbated on query $\mathcal{Q}_{s,f,m}$ with $|C_i| = 1M$. Here, the few results that satisfy best all 3 predicates featured in $\mathcal{Q}_{s,f,m}$ are better distributed using $DTB$. On Figure 2.9c, we represent the minimum score of the $k^{th}$ result among the results returned by reducers. These results support our observation: the score of returned results is higher when distribution is defined using $DTB$, while unnecessary results with lower scores are returned with $LPT$.

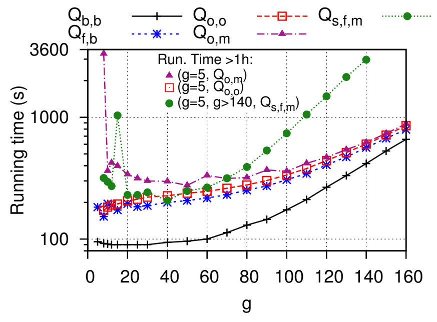
**TopBuckets Strategies.** We conduct experiments on the *TopBuckets* strategies exposed in Section 2.3.2.3. We vary the number of collections $n$ using queries $\mathcal{Q}_{b^*}$, $\mathcal{Q}_{o^*}$ and $\mathcal{Q}_{m^*}$.

Figure 2.10 summarizes the results. We do not report results where running time exceeds 1 hour. Experiments show the inefficiency of BRUTE-FORCE and TWO-PHASE. On these strategies, $n$-tuples of buckets, where $n \in 3 \ldots 5$, are assigned a score bound by the solver. With BRUTE-FORCE, the running time of *TopBuckets* quickly increases (solid black box on Figure 2.10) with $n$, because the solver needs to compute score bounds for a large number of bucket combinations, each one requiring to assign $2n$ variables. The TWO-PHASE strategy only beats BRUTE-FORCE on $\mathcal{Q}_{b^*}$ (Figure 2.10a) where its first phase prunes a large share (more than 99% for any $n$) of possible bucket combinations, thus limiting the running time of the second phase, that computes exact bounds using a smaller set of bucket combinations. On others queries, TWO-PHASE does not improve running time: the first phase does not prune enough combinations (e.g. 52% on $\mathcal{Q}_{m^*}$ for $n = 4$) to lessen the cost of the second phase where remaining combinations need to be assigned tight score bounds. The LOOSE strategy is the most efficient: (i) loose bounds do not impact significantly join processing time as a large share of potential results (e.g. 81% on $\mathcal{Q}_{o^*}$ for $n = 4$) remained pruned and (ii) *TopBuckets* scales with the number of collections $n$. In the remainder of our experiments, we use LOOSE as the *TopBuckets* strategy.

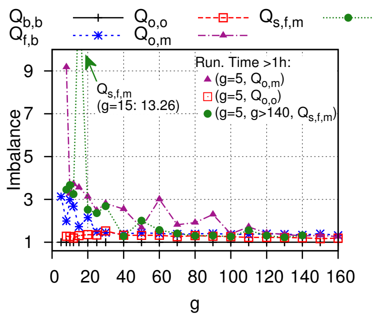
**Number of Granules.** Since the second phase of *TKIJ* relies on collected statistics to prune the input space and distribute the workload, we expect *TKIJ* to depend on the granularity of statistics (e.g. coarse or fine-grained). We conduct experiments to validate this intuition, varying the number of granules $g$.

We present respectively on Figures 2.11a, 2.11b and 2.11c the total running time of a range of queries, the load imbalance of the join phase computed using $\frac{\textit{Max Time Reducer}}{\textit{Average Time Reducer}}$ and the detailed running time for query $\mathcal{Q}_{o,m}$. We do not report results for executions where the total running time exceeds 1 hour. On queries that return the fewest high-scoring results ($\mathcal{Q}_{o,m}$, $\mathcal{Q}_{s,f,m}$), we observe on Figure 2.11a that with a lower $g$ (coarse statistics), running time degrades. *TKIJ* suffers here from poor workload distribution. This is expected as with fewer granules, we have fewer bucket combinations and especially fewer high-scoring ones. As *TKIJ* relies on a round-robin distribution of high-scoring buckets combinations, there is a lower probability to provide each reducer high-scoring results. Hence, we can observe on Figure 2.11b the imbalance that is more variable when $g$ decreases. As illustrated on Figure 2.11c, when $g$ increases, the local join processing is faster, thanks to a better workload distribution and to a larger pruning of unnecessary results. 81% of potential results are pruned for $g = 20$, while it is 96% for $g = 100$ (grey filled curve on Figure 2.11c). Yet, the pruning process *TopBuckets* is slower when $g$ increases and thus worsen the overall response time. Note that because it features more predicates, query $\mathcal{Q}_{s,f,m}$ requires to evaluate more bucket pairs during the *TopBuckets* process. Thus, *TopBuckets* running time increases faster with $g$ than on others queries, hence the impact on the overall response time.
For queries $\mathcal{Q}_{b,b}$ and $\mathcal{Q}_{o,o}$, coarse statistics have nearly no effect since a large number of high-scoring results can be found during the join phase (except when $g = 5$ where workload distribution fails for query $\mathcal{Q}_{o,o}$: a reducer does not quickly find high-scoring results). Finally, we observe that a number of granules $g \approx 40$ provides the best trade-off for the various queries that we experimented. Future investigations include the design of benchmark approaches or the adaptation of optimization techniques [35] to compute the optimal number of granules that minimizes execution time of *TKIJ*.

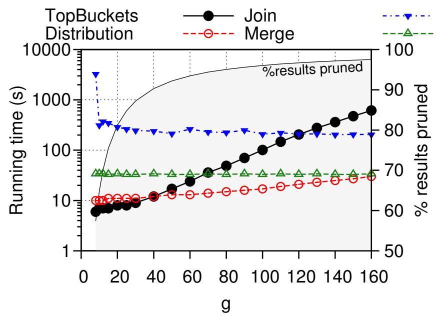
**Scalability.** We vary the size of collections $|C_i|$ to evaluate the scalability of *TKIJ*. We compare *TKIJ* to state-of-the-art competitors.
As a baseline, we borrow algorithms from related work on processing interval joins on Map-Reduce [29]. In this work, algorithms *RCCIS* and *All-Matrix* are designed to process interval joins using Allen predicates. In our settings, we use these algorithms to return only results that satisfy all the Boolean predicates of a RTJ query (i.e. a subset of top-$k$ results). We also impose reducers to stop join processing if $k$ results are found. Then, we merge and sort local results using a final Map-Reduce phase, identically to *TKIJ*.
*RCCIS* handles only *colocation* predicates where intervals intersect (e.g. *overlaps, meets*). *All-Matrix* handles only *sequence* predicates (*before, after*). *RCCIS* and *All-Matrix* also partition the temporal range using contiguous granules. For *RCCIS*, we set the number of granules to 24 which implies that 24 reducers are used. For *All-Matrix*, the number



(a) Running time



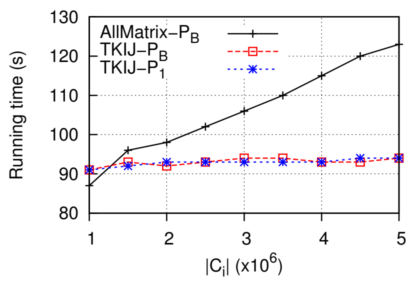
(b) Imbalance



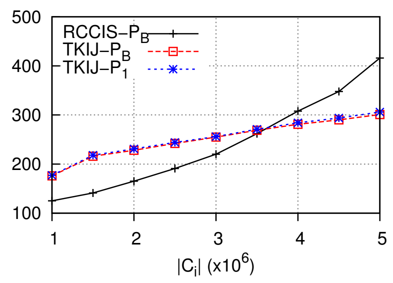
(c) Detailed running time for $\mathcal{Q}_{o,m}$

Parameters: $k = 100, |C_i| = 2 \times 10^6$, $\mathcal{P} = \mathcal{P}_1$, *TopBuckets*: LOOSE

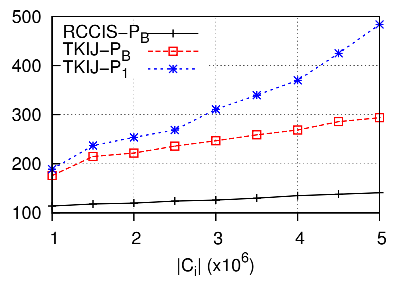
Figure 2.11: Experiments on synthetic data: effect of number of granules $g$



(a) $\mathcal{Q}_{b,b}$



(b) $\mathcal{Q}_{o,o}$



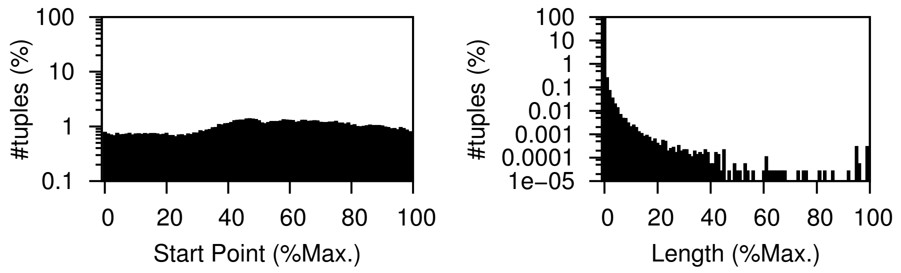
(c) $\mathcal{Q}_{s,m}$

Parameters: $g = 40$, $k = 100$, *TopBuckets*: LOOSE

Figure 2.12: Experiments on synthetic data: scalability

of reducers depends on the number of granules and of collections. We used 4 granules with $n = 3$, yielding 20 reducers. For *TKIJ*, we conduct a first set of experiments with score parameters $\mathcal{P}_B = \{(0,0),(0,0)\}$ (see Table 2.2). We are hence using a Boolean interpretation of predicates. Because *TKIJ* must return $k$ results, if only $k' < k$ results satisfy the Boolean predicates (with $\mathcal{S}(t) = 1.0$), $k - k'$ other results that do not satisfy at least one predicate will be returned (with $\mathcal{S}(t) < 1.0$). Thus, *TKIJ* may need to return more results than *All-Matrix* or *RCCIS*, that only return results fully satisfying all Boolean predicates. Hence, results of each algorithm are not directly comparable. For *TKIJ*, we conduct a second set of experiments using scored predicates with the score parameters $\mathcal{P}_1$.

**Results.** We present total running times on Figure 2.12. For query $\mathcal{Q}_{b,b}$ (Figure 2.12a), *TKIJ* remains nearly constant since *TopBuckets* returns only 1 bucket combination. Thus, *TKIJ* processes only a small share of input data. *All-Matrix* shuffles intervals belonging to all possible results, hence running time increases with $|C_i|$. For query $\mathcal{Q}_{o,o}$ (Figure 2.12b), *TopBuckets* selects more combinations to process. As the number of intervals per bucket increases with $|C_i|$, more data is shuffled and processed during the local join



(a) Start point (b) Length

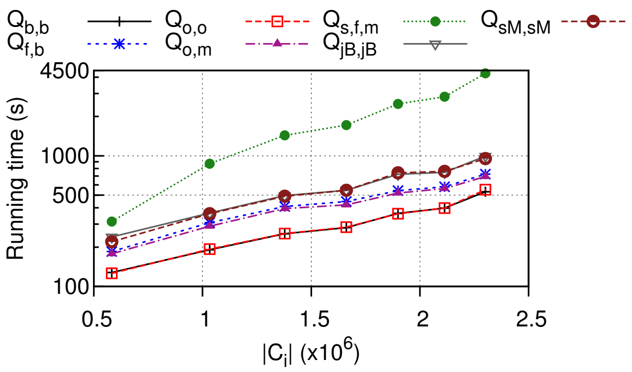
Figure 2.13: Network traffic data distribution

processing, hence the running time increases linearly with $|C_i|$. Figure 2.12b also shows that *TKIJ* outperforms *RCCIS* on $|C_i| > 3.5 \times 10^6$. In *RCCIS*, a first Map-Reduce phase builds intermediate results to determine which intervals need to be replicated to ensure output correctness in the join phase. Thus, its running time increases with $|C_i|$. Meanwhile, *TKIJ* decides which tuples should be combined on the basis of *TopBuckets*, which does not depend on $|C_i|$ and is on average 93% faster than the first phase of *RCCIS*. On $\mathcal{Q}_{s,m}$ (Figure 2.12c), we do not observe this phenomenon anymore: *RCCIS* first phase is faster (there are fewer intermediate results), while *TKIJ*'s join phase is longer (there are fewer high-scoring results). We observe a notable difference on query $\mathcal{Q}_{s,m}$ with scored and with Boolean predicates. The local join processing explains this difference. In the Boolean case, *TKIJ* focuses on building results where join conditions are satisfied (whose score is strictly positive), thus limiting the search space. With the approximate interpretation of predicates, more combinations need to be considered because tolerance on endpoints incurs a higher number of results with a strictly positive score. Thus, more intermediate results are computed. On query $\mathcal{Q}_{o,o}$, a large number of results have the highest score (Figure 2.8) which incurs major pruning. That justifies the absence of a significant difference between the scored and Boolean cases.

**Effect of k.** We conducted experiments where $k$ varies in $[10, 10^5]$ on a range of queries ($\mathcal{Q}_{b,b}$, $\mathcal{Q}_{o,o}$, $\mathcal{Q}_{s,f,m}$, $\mathcal{Q}_{f,b}$, $\mathcal{Q}_{o,m}$) with $|C_i| = 2\times10^6$. We observed that *TKIJ* is almost constant on all queries and all values of $k$. Actually, a large number ($> 10^{13}$) of potential results fall in each bucket combination. Thus, the set of selected bucket combinations remains the same for $k \in [10, 10^5]$ since we can always guarantee to return the correct top-$k$ results.

### 2.3.3.4 Network Traffic Data

**Data.** We use network traffic data collected on firewall logs of a data hosting company. Each log contains packets exchanged between servers and clients ($\approx$ 5GB, 100M packets per day). Each packet has a timestamp (seconds). We selected one log and built a list of connections by grouping packets exchanged between a pair (server, client). Only consecu-



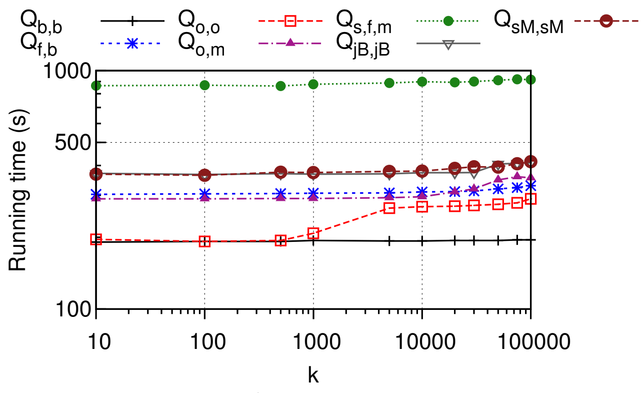
Parameters: $g = 40, k = 100, \mathcal{P} = \mathcal{P}_3$, *TopBuckets*: LOOSE

Figure 2.14: Experiments on network traffic data: scalability

tive packets whose timestamps are within a time interval $[0, 60]$ are grouped. A connection $[client, server, start, end]$ represents the activity of *client* on *server*, where the first packet was sent or received at timestamp *start* and the last one at timestamp *end*. The dataset obtained includes 3,636,814 intervals ($\approx$ 83MB), whose minimum, maximum, and average length are respectively 1, 86,459 and 54 seconds. Figure 2.13 shows the distribution of start points and lengths. Then, we copy each list of connections 3 times and process 3-way queries. We are interested in real-life scenarios occurring in network traffic analysis, hence we process queries $\mathcal{Q}_{jB,jB}$ and $\mathcal{Q}_{sM,sM}$ (Table 2.1). Query $\mathcal{Q}_{jB,jB}$ returns the sequences of connections that closely follow each other, while $\mathcal{Q}_{sM,sM}$ returns sequences where a delay was observed between two connections.

**Scalability.** We verify that *TKIJ* scales with various dataset sizes. When generating connections, we use various randomly selected samples on the log file used. We pick from 5% to 35% of input data. We obtain collections of connections whose number of intervals varies from $0.58 \times 10^6$ to $2.31 \times 10^6$.

We present total running times on Figure 2.14. We note that running time increases faster than what was observed on synthetic data. Here, when we use larger samples on input data, we have more buckets containing at least an interval. When $|C_i| = 0.58 \times 10^6$, there are 151 buckets containing at least an interval, while there are 296 for $|C_i| = 2.31 \times 10^6$. Thus, *TopBuckets* has to process more bucket combinations with higher $|C_i|$. On query $\mathcal{Q}_{s,f,m}$, that features more predicates, the time taken by *TopBuckets* is dominant (e.g. 82% of overall response time for $|C_i| = 1.38 \times 10^6$). Hence, overall response time increases faster on query $\mathcal{Q}_{s,f,m}$ than on all other queries. We also observe that while $\mathcal{Q}_{o,o}$ lasts longer on synthetic data, *TKIJ* performs similarly on $\mathcal{Q}_{b,b}$ and $\mathcal{Q}_{o,o}$ on real data. That can be explained by the fact that the real dataset contains long intervals (Figure 2.13). These intervals fall into buckets built with granules that are far apart (e.g. $b_1 = ([2160, 4320], [19440, 21600])$ or $b_2 = ([8640, 10800], [38880, 41040])$). Thus, we can find bucket combinations (e.g. $\omega = (b_1, b_2)$) whose results $(x, y)$ are guaranteed to have a high



Param.: $|C_i| = 1.03 \times 10^6, g = 40, \mathcal{P} = \mathcal{P}_3,$ *TopBuckets*: LOOSE

Figure 2.15: Experiments on network traffic data: effect of $k$

score *s-overlaps*$(x, y)$. Then, *TopBuckets* returns less bucket combinations, reducing the search space while guaranteeing correctness.

**Effect of k.** We verify *TKIJ* on various values of $k$. We present running times on Figure 2.15. For all queries except $\mathcal{Q}_{o,o}$, we observe that *TKIJ* remains nearly constant when $k \leq 5000$. Then, the running time increases slowly when $k > 5000$: as more results need to be returned, more intermediate results are built before termination especially with queries having fewer high-scoring results. On $\mathcal{Q}_{o,o}$ we observe that *TKIJ* increases slightly between $k = 1000$ and $k = 5000$. That is explained by an increase (from 643 to 41,272) in the number of bucket combinations $|\Omega_{k,\mathcal{S}}|$ necessary to return the correct top-$k$ results, which in turn increases the number of intermediate results.

### 2.3.4 Related Work on Batch Temporal Joins

Three research areas relate to our work, however none of them addresses the RTJ problem.

#### Interval Joins

The closest work to ours [29] addresses the processing of multi-way joins on Map-Reduce for Allen Boolean predicates [7]. The first algorithm, *RCCIS*, solves *colocation* queries, where all predicates require intervals to have a non-empty intersection (e.g. *overlaps*, *meets*). *RCCIS* reduces the amount of data shuffled in Map-Reduce by sending to the same reducer intervals that are most likely to be collocated and produce a join result. The second algorithm, *All-Matrix* handles *sequence* queries. Because such queries imply unavoidable replication, *All-Matrix* focuses on load balancing. Chawda et al. [29] also design algorithms to evaluate hybrid queries that feature combination of colocation predicates, sequence predicates or use interval attributes other than interval bounds. In a centralized setting, Gao et al. [54] investigated various join algorithms. They handle a variety of join predicates, including Allen's predicates. None of those algorithms is applicable to solving the RTJ problem as they do not handle scored results.

Several studies investigated efficient processing of queries involving the *intersects*[2] predicate [35, 44, 78, 100, 113, 132]. The idea is to combine events that share a common period of validity. A recent investigation proposed a compound index structure using segment trees to find intervals that *intersect* a query-interval in a key-value cloud-store [113]. Kaufmann et al. [78] designed a space-efficient index that allows to optimize the evaluation of a range of temporal queries, including temporal aggregation or temporal joins. This index can be efficiently compressed, which allows to query large amounts of temporal data using in-memory databases. Dignös et al. [35] focused on optimizing partitioning of the time range to improve join processing using the intersects predicate. They assume the time range is uniformly partitioned into *granules.* A partition is a sequence of one or more adjacent granules and an interval is assigned to the smallest partition that contains this interval. Partitioning is leveraged to evaluate efficiently intersection joins: only intervals from some combinations of partitions have to be joined. Dignös et al. focused on limiting the number of *false hits* (when a partition is fetched, but does not yield join results) and *partition accesses.* They propose an approach to find the optimal number of granules, that maximizes the efficiency of join processing. Recently, Piatov et al. [100] focused on designing an efficient data structure for intersection joins for in-memory processing. The goal is to have a structure that is as compact as possible (that fits in cache/memory) and optimizes CPU usage. A series of work on spatio-temporal data also focused on efficiently retrieving *overlapping* objects with Boolean semantics. Objects are stored in partitions [35, 88], or tree structures [44], such as the R-Tree [14, 61] or the quadtree [52]. All these studies are not applicable to our settings since we focus on richer semantics, including an approximate interpretation of Allen's *meets* or *before.*

A line of work in approximate reasoning investigated flexible interpretations of Allen predicates [41, 96, 111]. Dubois et al. [41] propose a flexible approach for scoring Allen predicates. These approaches [41, 96, 111] both support a *crisp* interpretation of intervals, where the intervals bounds are assumed to be known and exact, and a *fuzzy* interpretation, where an interval is represented using a fuzzy set. These studies only focused on semantics and none of them designed join algorithms.

Snodgrass proposed the TSQL2 temporal query language [117]. TSQL2 includes a set of boolean operators that allow to compare *periods*, *datetimes* and *intervals* (here, unanchored periods of time). Since (i) the Snodgrass model subsumes Allen's [117] and (ii) Allen's temporal predicates are used in a recent work on distributed evaluation of interval joins [29], we started our investigation with Allen's predicates. We could extend our work to support the flexibility of Snodgrass (e.g., supporting predicates that compare *datetimes*).

[2]In some studies [35, 44, 100, 113], "intersects" is also known as "overlaps", although it does not has the same semantics as Allen's "overlaps" [7]. In our work, we use Allen's semantics.

## Top-k Processing

In *top-k selection* queries [73] the score of an object is given by a function that aggregates objects' base scores over several dimensions and the query returns the $k$ objects with the best score (e.g. the highest). Fagin et al. [45, 46] developed several notorious *instance-optimal* algorithms to evaluate efficiently top-$k$ selection queries, that avoid evaluating unnecessary results.

These investigation gave rise to another line of work: *rank-join*[3] processing [37, 51, 71, 72, 94, 95, 110]. Here, the query returns the top-$k$ *join results* and the score of a result is the aggregation of objects' individual scores. Rank-join queries are usually expressed in SQL using a `STOP AFTER` clause, that was proposed in the seminal work of Carey and Kossmann [23]. Ilyas et al. [72] designed *HRJN*, an instance-optimal algorithm. They assume that we can access input relations in descending of score. This algorithm avoids computing unnecessary results and relies on the monotonicity of the aggregation function to terminate early. This work led to the design of *Pull-Bound Rank Join* algorithms [51, 110] that focused on obtained better guarantees on I/O costs. All these studies rely on the fact that objects' scores are known *a priori*. In our settings, these scores are *predicate-dependent*, which make these techniques inapplicable.

Some studies investigated distributed or *highly distributed* settings [37, 95, 126]. Specifically, in NoSQL databases, the *BFHM* algorithm [95] places each tuple in a bucket that depends on its score. Tuples are compressed using Bloom-filters allowing to select the *best* buckets first then retrieve tuples to be joined from the database. Because a Bloom-filter yields false positives, *BFHM* may require several iterations. Our work differs in two aspects. First, this study assumes that the score are known *a priori*. Second, reiterating join processing in our case would incur too high an overhead.

Chang and Hwang [27] studied how to minimize the number of *expensive ranking predicates* that need to be evaluated to return top-$k$ results. An *expensive ranking predicate* may be a user-defined function that gives the score of a pair of objects (e.g. how close are two objects). They study *fuzzy joins* that combine objects from several collections. A distinctive feature of their work is that scores are hence *predicate-dependent*. However, our work has several differences. First, they assume that at least one *cheap* ranking predicate allows a sorted access, which enables optimizations in their algorithm. We do not assume the existence of such predicate. Second, they essentially focus on a centralized setting. They propose an extension where data is partitioned, but it requires several iterations during top-$k$ processing, which is a too high overhead in our context.

Li et al. [85] investigate a special kind of top-$k$ queries on temporal data. Here, the score of objects are given by *score functions*, that evolve over time. This is useful when the data that is monitored is modeled using piecewise linear function that approximate real measures (e.g. a stock price, sensor readings). Their query returns the top-$k$ *functions* and hence differs from our work.

[3]Also known as *top-k join* [73].

### Distributed Join Processing

Because it is an expensive operation, join has been extensively studied in distributed settings. This raises well-known challenges, such as load balancing or limiting replication. RanKloud [20] computes data statistics to retrieve an estimation of the *kth* join score. Then, only the part of input data whose score is above the estimated one is uniformly distributed and processed in parallel on a set of workers. Similarly, we rely on computing statistics to avoid processing useless data. However, while RanKloud outputs approximate results, our algorithm guarantees to return exact top-$k$ results. Another work [36] proposes a partitioning scheme on Map-Reduce based on partitioning for parallel skyline query processing [125]. The *angle-based partitioning* distributes evenly the volume that contains points near the best possible point, increasing the probability that skyline points will fall evenly in each partition. This idea is reintroduced in the context of top-$k$ joins [36] and is shown to be superior to a cardinality-based partitioning. Although we share the same intuition, we cannot directly apply this technique since it assumes that partial scores are known *a priori* while in our case they are predicate-dependent.

Since it was introduced by Dean and Ghemawat [34] the Map-Reduce paradigm and its open-source implementation Hadoop Map-Reduce became popular for distributed processing on a cluster of (commodity) machines. Although its relevance is questionable on some applications [118], many studies focused on join processing using Map-Reduce [5, 17, 29, 97, 134], including $k$-nearest neighbor joins [89] or similarity joins [81]. Okcan and Riedewald [97] proposed to use matrix model to optimize load balance for 2-way joins. Zhang et al. [134] partitioned data using an hypercube model and leveraged the properties of Hilbert curves to optimize both load balance and replication for multi-way joins. Afrati et al. [4] focused on limiting replication for multi-way equi-joins. All these studies are not directly applicable to our setting and focus on different query models that do not leverage temporal semantics.

## 2.4 Top-k Temporal Joins: Stream Processing

In this section, we present a *preliminary study* for monitoring temporal data using stream processing. We present detailed investigation directions that are a basis for designing a distributed query evaluation approach for top-$k$ temporal stream joins.

### Motivation

Our work is a direct extension of our investigations on batch processing of top-$k$ temporal joins. Batch processing is best suited to applications where all data is given at once and a response time in *minutes* is acceptable. In this context, we evaluate queries over a *fixed dataset*. We showed how an analyst could collect data from firewalls and query large datasets using our evaluation approach *TKIJ*. However, in some applications, including in network traffic monitoring, an analyst also needs to query data as it arrives, in the forms of *streams*, and she requires a response time in *seconds* or *sub-seconds.* For instance, *TKIJ* runs in a single round with three Map-Reduce jobs and collects statistics on the *whole* dataset to guarantee the correctness of results. Repeating top-$k$ processing to update statistics and results would require a high number of I/O accesses, which is expensive. Moreover, it returns results in *minutes* for large datasets, and it is based on an implementation on Map-Reduce that is best suited for batch processing and ETL operations [118]. This motivates the need for investigating stream processing of top-$k$ temporal joins. In this setting, we aim at evaluating *continuous queries* that are continuously returning *most recent* results using *continuous datasets.*

Previous studies on stream join processing has considered the notion of recency in two main flavors. Some studies advocated the need to process *full-history* joins [43, 87]. In this setting, all input tuples are joined with tuples that have already arrived and are stored on the system. This is best suited to applications that "require maintaining large historical states" [43]. Other studies considered *window-based* joins [58, 60, 76, 77, 120, 123, 127]. Here, queries are continuous and they operate on continuous datasets. We propose to study window-based joins with a *time-based* window: only tuples that have a *recent timestamp* are valid. This approach is compatible with the requirement of temporal data monitoring and top-$k$ join processing: we will return the top-$k$ *most recent* results.

## Challenges

We aim at designing a distributed query evaluation approach. In this context, stream join processing raises specific challenges. First, a distributed query evaluation approach must ensure load balancing and limit replication. Streams are by nature dynamic and "possibly unpredictable" [12], which requires to build an approach that adapts to the stream dynamics. Previous studies have investigated various approaches to tackle this problem [43, 87], however they focused on 2-way joins using hash or random partitioning. These techniques are not applicable in our setting where we aim at joining more than 2 input streams. Second, we aim to minimize system *latency* which is a natural requirement for monitoring data in real-time. We can define latency as the difference between the time an input tuple enters the system and the time at which the first corresponding join result is returned [43](i.e. added to the top-$k$ results). In our setting, because of the approximate semantics, every combination of intervals from different input streams is a potential join result. However, we aim at returning the correct top-$k$ results, based on the whole data. Moreover, because of the window semantics, a result that was not included in the top-$k$ results at time $t_1$ may need to be included later at time $t_2 > t_1$. This enforces maintaining the top-$k$ results using results that need to be formerly stored. Therefore, at any time $t$, a large number of interval combinations are potentially in the top-$k$ results. To minimize latency, we need to (i) avoid evaluating unnecessary results and (ii) efficiently store and maintain the potential top-$k$ results. This can be achieved by leveraging the top-$k$ semantics. As a summary, we need to devise an efficient query evaluation approach that is able to quickly ingest input tuples and update top-$k$ results.

## 2.4.1 Data Model and Problem for Stream Temporal Joins

### 2.4.1.1 Data Model

We are given $m$ streams $S_1, \dots, S_m$. A stream $S_i$ is a continuous, unbounded sequence of intervals denoted $x_i$. Each interval $x_i \in S_i$ has a unique identifier $x_i.id$, a start time $\underline{x_i}$ and an end time $\overline{x_i}$.

Our model assumes that the sources of streams $S_i$ are able to extract interval bounds based on raw point-based events. Such a process can be achieved either by low level physical sensors with sufficient computing resources, either by middlewares. This is a reasonable assumption that covers a wide range of scenarios. For instance, in network traffic monitoring, a firewall monitors packets exchanged between servers and clients in the form (*timestamp, client.IP*, *server.IP, portNumber*). A middleware could monitor clients connected to the server and emit a tuple (*client.IP&server.IP*, $t_1, t_2$) that represent the client connection during interval $[t_1, t_2]$. A connection would be obtained by grouping consecutive packets exchanged between a client and a server. The fact that streams emit intervals was also assumed by Li et al. [86] who proposed to support *interval events* in the domain of Complex Event Processing (CEP). *To the best of our knowledge, no previous study on stream join processing focused on supporting temporal intervals.*

### 2.4.1.2 Stream Ranked Temporal Join

We are interested in expressing $m$-ary join queries on streams $S_1, \dots, S_m$. Each stream $S_i$ is associated a window size $W_i$. The sliding window on stream $S_i$ is coined $S_i[W_i]$. To the best of our knowledge, all previous studies that used time-based windows assumed that each tuple has a *single* timestamp. In our setting, streams feature *temporal intervals*, thus we need to devise proper semantics. At time $t$, we consider that $x_i \in S_i[W_i]$ if and only if $\overline{x_i} \in [t - W_i, t]$. This definition implies that an interval $x_i$ is valid if it was *recently completed.* This is consistent with with previous approaches on query processing on interval data [35, 44, 113] that considered a tuple $x_i$ to be valid at time $t$ when $t \in [\underline{x_i}, \overline{x_i}]$. Therefore, our model is based on the *application* timestamp[4] $\overline{x_i}$ that is given by stream sources [12]. This approach contrasts with the assignment of a *system* timestamp[5] to a tuple when it arrives [60, 87, 91].

We adopt the semantics of window joins made explicit by Ji et al. [76]: when a new interval $x_i \in S_i$ arrives, (i) we invalidate expired intervals in all windows $S_j[W_j]$ where $j \neq i$ using the timestamp of $x_i$ and (ii) we produce *valid results* using valid input intervals. We

[4]Also known as *explicit* timestamp.

[5]Also known as a *implicit* timestamp.

denote *valid result* a tuple $(x_1, \ldots, x_m) \in S_1 \times \ldots \times S_m$ where each $x_i$ is valid at some time $t$. The procedure above implicitly defines the semantics of a valid result [76]:

*Valid Result.* A result $(x_1, \ldots, x_m)$ is valid with respect to sliding windows $S_i[W_i]$ $i \in 1, \ldots, m$ if $\forall i, j \in 1, \ldots, m$ , $j \neq i :$ $\overline{x}_j - W_i \leq \overline{x}_i \leq \overline{x}_j + W_j$.

This definition ensures that if $x_i \in S_i$ arrives before (resp. after) $x_j \in S_j$ where $j \neq i$, then $x_i \in S_i[W_i]$ (resp. $x_j \in S_j[W_j]$). We assume that streams have no *intra-stream* nor *inter-stream disorder* [76]: intervals arrive in non-decreasing order of timestamp. How to handle stream disorder is orthogonal to our work. We assume a *lazy* evaluation: new query results are returned each time a new interval arrives[6]. This is a common assumption in stream join processing [76, 77, 120, 127].

Then, we extend RTJ queries to define a Stream Ranked Temporal Join query (S-RTJ). We adapt the model developed for RTJ queries (Section 2.3.1). We model S-RTJ queries using a graph, where vertices are streams and edges are labeled with scored temporal predicates. The score of each tuple is computed using a function $\mathcal{S}$ that aggregates the partial scores assigned by each predicate. The S-RTJ problem is thus the following:

*S-RTJ Problem.* At any time, evaluating a S-RTJ query requires to return a top-$k$ set of *valid results* of the form $(x_1, \ldots, x_m) \in S_1 \times \ldots \times S_m$ ranked by (descending) order of $\mathcal{S}_{(i,j) \in E}(s\text{-}p_{(i,j)}(x_i, x_j))$.

We aim to evaluate continuously S-RTJ queries: when a new interval arrives, (i) we invalidate expired intervals and results that include expired intervals and (ii) we return the current S-RTJ results.

## 2.4.2 Preliminary Study for Processing Stream Temporal Joins

### 2.4.2.1 Overview

We propose to adapt *TKIJ*, our approach for batch processing of RTJ queries. This new approach is based on 4 main components shown in Figure 2.16. A first phase (a) collects statistics. It uses the same idea as *TKIJ* (Section 2.3.2.2): partitioning time into granules and collecting the number of intervals that fall into each bucket (a pair of granules). This phase is executed online, as data arrives continuously. A second phase (b) leverages these statistics to prune unnecessary results. This is similar to *TKIJ* approach (Section 2.3.2.3). This phase has to leverage the time-based window semantics to determine if a bucket can be invalidated. For instance, given a query, we may decide if intervals from a bucket can be safely invalidated if we are sure that they will *never* be included in top-$k$ results. A third phase (c) distributes the workload on a set of processing nodes. It leverages the collected statistics and the results of the previous phase. It decides on which processing node(s) a bucket has to be sent so as to evaluate locally a full S-RTJ query (d). Finally, on notable difference with *TKIJ* is that we can transmit the score of the $k$-th result to the pruning

[6]An *active* evaluation would compute results at any time or at a given period.

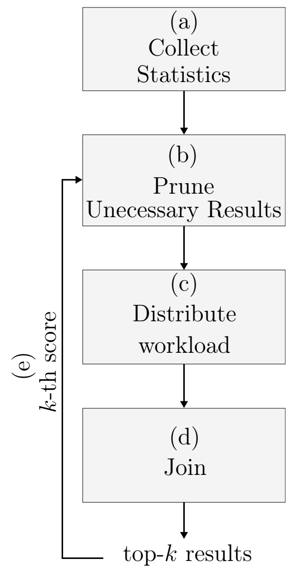


Figure 2.16: Overview of distributed stream top-$k$ join processing

phase (e) since we are evaluating results continuously. The rationale is to use this score to prune more results: only results with a potential score above the $k$-th score need to be evaluated.

#### 2.4.2.2 Future Investigations

We identified several investigation directions to tackle our two main challenges.

**Distributed Processing.** Our first challenge is to optimize load balancing and data replication in the context of stream join processing.

- **One-hop vs multi-hop routing.** Previous studies have investigated different routing schemes to compute join results in a distributed settings. Gu et al. [60] proposed two routing schemes for distributed join processing. A first scheme, *ATR* is a *single-hop* scheme where no intermediate join results are passed between nodes. A processing node thus receives input tuples from all streams and evaluate a full join. A second scheme, *CTR* is a *multi-hop* scheme, where intermediate join results are passed between nodes. Each processing node evaluates a partial join, and emits intermediate results or full join results. Gu et al. conducted experiments showing that *ATR* performs better than *CTR* when join selectivity is high. Later, Wang and Rundensteiner [127] designed a multi-hop scheme that outperforms both *ATR* and *CTR*. Zhou et al. [136] proposed an algorithm that chooses the best routing scheme based on streams arrival rate. In our settings, we advocate for building a single-hop scheme. Our query naturally returns a high number of intermediate results, since any combination is a potential result. This is akin to a query that has a *high selectivity.*

Zhou et al. [136] showed that when join selectivity is high, their optimized plan was equivalent to this plan, which confirms the need for a single-hop scheme.

- **Workload distribution.** To optimize load balance and limit replication, we can adapt the idea developed for *TKIJ*: leverage collected statistics. In this approach, intervals are grouped by buckets and all intervals from the same bucket are sent to the same node(s). The distribution uses in input the bucket combinations that were not pruned by the pruning phase and ensures that each combination will be evaluated on a processing node. A first naive approach is to evenly distribute buckets from one stream and map buckets from others streams to the required processing nodes. This is akin to using random partitioning for intervals from one input stream, and then replicating intervals from other streams where needed. This approach incurs a lot of data replication and a high communication overhead. Moreover, it is not adaptive. A smarter approach has to adapt workload distribution to the dynamics of input streams. We aim to investigate further in this direction.

- **Data Storage.** Because of the window semantics, we also need to store temporarily intervals that could be joined later with other intervals. An interval that arrives in the system has (i) to be joined with previously arrived intervals that were not expired and (ii) to be stored and joined with intervals arriving afterward. Intervals need to be stored in-memory to maximize efficiency. They are later discarded when they expire. Designing a distributed system that limits replication while storing only necessary input tuples and guaranteeing correctness is a tedious task. Lin et al. [87] proposed a biclique model for 2-way joins: an input tuple is stored on one side of the biclique (each graph node is a processing node) and sent to the other side to join with intervals from the other relation. Adapting this approach to $m$-way joins is not straightforward. We need to devise an approach that leverages the top-$k$ and window semantics to store (and discard) intervals efficiently.

**Latency.** Our second challenge is to minimize latency, which is tedious since every combination is a potential join result in our settings.

- **Pruning Unnecessary Results.** Pruning unnecessary results is a key component of our approach. For batch processing, we rely on a constraint programming solver to compute score bounds of a bucket combination. Then, we leveraged these bounds to prune unnecessary bucket combinations (and thus unnecessary results). In our experiments on *TKIJ*, we observed that the solver's response time could be high if a large number of bucket combinations were considered. This was acceptable in batch processing, where the overall response time may be in minutes. This is not satisfactory for a low-latency system where latency should be in seconds or sub-seconds. One possible improvement is to reuse previous computations. Indeed, given that our predicates only compare the *relative* position of interval bounds, the score bounds of a bucket combination only depend on the relative position of buckets. Therefore, the score bound of a combination $\omega_1 = (b_{1,1,1}, b_{2,1,1})$ may also be valid for another combination $\omega_2 = (b_{1,5,5}, b_{2,5,5})$. Here, $\omega_1$ would concern intervals arrived earlier than those in $\omega_2$. This happens when uniform partitioning of the time range is used: the score bounds are similar on similar bucket combinations. We need to leverage this idea to reduce the usage of the solver and thus prune unnecessary results more efficiently.

- **Local Join Processing.** Even if workload if perfectly distributed, the latency still depends on the evaluation of full S-RTJ queries on processing nodes. First, we need to build efficiently join results. Many previous studies have investigated centralized stream join processing [58, 77, 123]. We need to adapt techniques to our temporal semantics. Second, how to efficiently maintain top-$k$ results is not a straightforward task given the time-based window semantics. A first approach could be to use the $k$-skyband to store results. Then, we can efficiently compute top-$k$ results using the $k$-skyband. We could adapt previous work on continuous top-$k$ queries [92, 115].

### 2.4.3 Related Work on Stream Temporal Joins

Three main research domains are related to our work. However, none addresses the S-RTJ problem.

#### Stream Join Processing

Stream join processing has been extensively studied, especially in centralized settings [58, 63, 77, 123]. Viglas and al. [123] showed the superiority of a single multi-way join operator over a tree of binary join operators. Golab et al. studied [58] several multi-way join algorithms with sliding window and proposed an heuristic to optimize join ordering based on stream arrival rates. Kang et al. [77] also proposed a cost model for 2-way joins and investigated different combinations of join algorithms for each input. Our investigations are orthogonal to these studies. We mainly focus on challenges induced by our distributed setting. We could adapt these studies for the *local* processing of S-RTJ queries.

Distributed stream join processing was also investigated [43, 60, 87, 99, 127, 136]. Gu et al. [60] proposed two routing schemes to distribute join processing over a set of nodes. To partition input tuples, they group all tuples from a given stream that arrived within a certain time period. The *ATR* scheme selects one input stream as the *master* stream, distributes time segments for this stream and aligns *slave* streams with the master. This is a *single-hop* scheme that does not transfer intermediate results between nodes. The *CTR* scheme dynamically routes tuples to a set of least-loaded host. It uses an heuristic to find a minimum set of nodes to achieve minimum overhead. This is a *multi-hop* scheme since intermediate join results are transferred between nodes.

Wang and Rundensteiner [127] proposed the *PSP* model that *slices* "*window states of a join operator into fine-grained window slices*". In the PSP model, both tuples and intermediate results are passed between nodes. A probe tuple is passed through a ring structure to purge expired tuples. A build tuple is used to construct join results. Intermediate results are passed until being dropped, after completing a loop. They also study load balancing, using workload smoothing and state relocation, and show the superiority of PSP over ATR or CTR. Although PSP does not focus on temporal joins nor on ranked semantics, it is a baseline query evaluation approach that handles any join predicates and thus could be compared to our approach for S-RTJ processing.

Studies have also considered load balancing and data replication, that are natural concerns in a distributed setting. Elseidy et al. [43] proposed an adaptive 2-way join operator that repartitions and relocate input data on a cluster. They use the notorious *join-matrix* model to partition data, and design a migration algorithm with performance guarantees. Lin et al. [87] designed the *biclique* model where processing nodes are organized using a biclique (a complete bipartite graph). Each side of the biclique corresponds to a relation, where tuples are stored. Random routing or hash partitioning are used to route input tuples. These studies are not applicable to our setting as they focus only on 2-way joins and it is not straightforward to adapt these techniques to multi-way joins.

A number of systems have been developed to handle streaming data, including Borealis [2], Flink [21], Heron [83], Photon [9], Spark [131], Squall [124], Storm [121] or TimeStream [105]. While most studies focus on general-purpose systems, some of them especially investigated join processing. Squall [124] includes join operators that leverage the *Hybrid-Hypercube* partitioning scheme. It is based on a combination of hash and random partitioning. In our settings, any combination is a potential join result, therefore it would require to partition one collection and replicate the others, which would not be efficient. Photon [9] was developed by Google to join streams of events (clicks and queries during web search). A strong emphasis was put on exactly-once semantics, fault-tolerance and latency. It relies on a registry that stores the id of events that have already been joined. It does not aims at supporting any kind of join predicate. In summary, these systems focus on supporting generic join queries and do not aim at tackling the specific challenges raised by our semantics.

### Top-k Queries

In the *top-k monitoring* domain [13, 129], a set of sensors, that are distributed over a network, emit readings (e.g. temperature, pollution, network traffic measures). One may be interested in retrieving the top-$k$ objects with the highest scores. For instance, Babcock and Olson [13] consider that each sensor emits a reading for a given object and that the object score is the sum of these readings. In these studies, a coordinator node fetches data from monitoring nodes and computes the top-$k$ results. The main challenge considered is to minimize the communication cost and to avoid querying too often sensors since this may drain their batteries. This differs from our setting, where data processing is distributed, hence these algorithms are not applicable for S-RTJ queries.

In the domain of *top-k queries over sliding windows* [92, 115], objects arrive in streams and studies aims at returning the top-$k$ objects that are in the sliding window. Mouratidis et al. [92] proposed an algorithm that leverages a skyband to return top-$k$ results. One important remark is that the $k$-skyband contains the correct top-$k$ results at any moment. Shen et al. [115] aim at returning the top-$k$ *pairs* of objects, using any function to aggregate objects' scores. They also use a skyband to store object pairs. They propose algorithms to (i) efficiently compute top-$k$ pairs using the skyband and to (ii) efficiently maintain the skyband as objects arrive or expire. Although these studies are closely related to our investigations, their techniques are not applicable to our setting. First, Mouratidis et al. [92] aggregate the *base score of objects*, assuming they are known *a priori*. In our settings, scores of results are *predicate-dependent* and thus are not known a priori. Shen et al. [115] aggregate the *score of objects*, therefore they do not rely on the fact that scores are known a priori. However, they do not focus on limiting the number of object pairs that are built: this would cause too high an overhead in our context.

### Complex Event Processing

Complex Event Processing (CEP) aims at filtering and combining *notifications of events* to build complex events that help understanding real-world phenomenons [32]. In this domain, a generic representation of an event is a tuple $e = \langle s, t \rangle$ where $s$ is a list of content attributes and $t$ a list of time attributes that denote occurrences of $e$ [53]. These events are often emitted by physical sensors, and CEP helps monitoring complex events that are combinations of events. A popular query model is proposed in SASE [128], whose structure is:

```
EVENT <event pattern>
[WHERE <qualifications>]
[WITHIN <window>]
```

The `EVENT` clause describes the event pattern that is required. For instance, a sequence of events of different types may be required. The `WHERE` clause filters events based on their attributes, and is similar to the homonym in SQL. The `WITHIN` clause specifies the time window during which combined events must occur. There has been a lot of research in CEP [6, 128] including in distributed settings [112], on supporting interval-based events [86] or imprecise timestamps [133]. CEP has natural similarities with stream processing. Cugola et al. [32] propose an overview of both domains and a framework to reunite them. We position our work in the domain of stream processing. We aim to *transform* input streams since we aim at returning top-$k$ results, that can be reused later. This is a distinctive feature of stream processing [32]. Moreover, stream processing studies have strongly focused on tackling challenges raised by join processing, whereas it is not an explicit concern in

CEP systems. However, we notice that two studies in CEP are closely related to ours in terms of semantics. Zhang et al [133] handle events with imprecise timestamps. They assume that an event has a probability to occur in a given uncertainty interval. This allows to tackle a number of challenges raised by imprecise measures or synchronization problems in distributed settings. However, they do no aim at supporting events that are characterized by intervals. Li et al. [86] focused on using interval-based events rather than point-based events. They propose an interval-event sequence operator that supports Allen's predicates. However, they do aim at handling an approximate interpretation of temporal predicates and do not support ranked semantics.

## 2.5 Conclusion

In this chapter, we investigated temporal data monitoring. We introduced a new kind of interval join, that features scoring functions reflecting the degree of satisfaction of temporal predicates.

First, we studied these joins in the context of batch processing [103]: we formalized Ranked Temporal Join (RTJ), that combine *collections* of intervals and return the $k$ best results. We designed *TKIJ*, a distributed RTJ query evaluation approach on Map-Reduce. *TKIJ* is a multi-way top-$k$ interval join algorithm that relies on an offline collection of statistics to prune the search space and a workload distribution scheme appropriate to top-$k$ processing. We conducted experiments on synthetic data that validate our approach and showed the efficiency of *TKIJ* on various queries. We observed the same effectiveness of *TKIJ* on real network traffic logs.

Second, we proposed a *preliminary study* to extend our work to stream processing. We modeled S-RTJ queries, an extension of RTJ queries to stream processing, that combine intervals arriving as *streams*. We exposed the challenges raised by distributed stream processing of S-RTJ queries and we outlined investigation directions to tackle these challenges.

# Chapter 3

# Motivation-Aware Task Assignment

In Chapter 1, we introduced adaptive task assignment in crowdsourcing. In practice, adaptive task assignment was proposed to tackle the challenges raised by crowdsourcing that can't be addressed by self-appointment of workers to tasks (e.g. low crowdwork quality). Interestingly, adaptive task assignment can be interpreted as the monitoring of workers' activity since it requires to monitor task completion in order to improve task assignment. We also introduced the need to incorporate worker motivation in task assignment: we aim to study *motivation-aware task assignment.*

In this chapter, we first present our proposal for motivation-aware task assignment. Then, we present our model and our investigations on two variants of motivation-aware task assignment: Individual Task Assignment (Ita) and Holistic Task Assignment (Hta).

## 3.1 Our Proposal: Motivation-Aware Task Assignment

Existing literature has extensively studied how to perform task assignment to workers on crowdsourcing platforms [47, 67, 68, 106, 109]. Task assignment considers goals such as maximizing the quality of completed tasks, or minimizing task cost and latency to complete tasks. More recently, some research has reported noticeable improvement in task outcome quality when human factors, such as workers' skills and expected wage, were used in assigning tasks to workers [106, 109].

Yet, even when tasks are perfectly matched and assigned to workers initially, an important longstanding problem is *how to keep motivating workers who are not well-engaged in completing assigned tasks.* Moreover, studies have shown that crowdsourcing platforms should account for the "*dynamic nature of motivation*" and "*support workers' diverse motivations* "[82] and that research efforts, mostly driven towards requesters' requirements, should be driven towards workers' needs [90]. To address this problem, some existing work focused on incentivizing workers for long-lasting tasks [25, 69] or entertaining workers during task completion [33]. Moreover, recent studies have experimentally demonstrated the importance of intrinsic motivation in task completion [108].

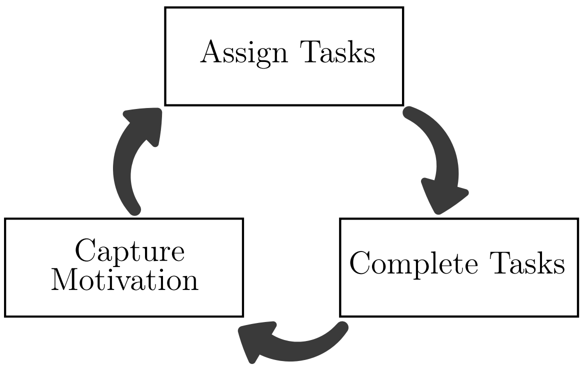


Figure 3.1: Overview of motivation-aware task assignment

## Proposal

While effective to some extent, previous studies do not perceive task completion as an iterative process within which workers' motivation evolves, neither do they model that in the task assignment process. Therefore, it becomes increasingly important to *understand and model workers' motivation appropriately in the task assignment step*. We advocate the need to *account for the evolution of workers' motivation* as workers complete tasks and *incorporate motivation in task assignment*.

In this thesis, we propose to capture worker motivation and leverage it in an iterative, adaptive task assignment process. Figure 3.1 illustrates our idea. At each iteration, we aim to assign tasks to a worker, let her completing tasks, and then capture her motivation. In the next iteration, task assignment leverages worker's motivation to assign motivating tasks.

## Scope

We study two natural variants of motivation-aware task assignment. The first one, **Individual Task Assignment** (Ita), *individually* assigns tasks to *one worker at a time*. The second one, **Holistic Task Assignment** (Hta), assigns tasks to *all available workers, holistically*. In each variant, we study different models to capture workers' motivation. In Ita, we see motivation as a combination of *task diversity*, that quantifies how different tasks are from each other, and *task payment*, that captures how well tasks pay. In Hta, we combine *task diversity* and *task relevance*, that captures how proficient a worker believes to be for a task.

### 3.1.1 Motivation-Aware Task Assignment: Challenges

Our first challenge is to model a worker's motivation. While some workers may be driven by fun and enjoyment, others may look to advance their human capital, or increase their compensation. In fact, there are more than 13 factors that could be used to model motivation according to [79] (e.g., *task payment*, *task diversity*, *task autonomy*, *task identity*, *human capital advancement*, *pastime*). In addition, in a given work session, a worker's motivation for a task may also depend on tasks that she has already completed and on other available tasks.

Our second challenge, is to formulate motivation-aware task assignment. Previous studies on adaptive task assignment did not include explicitly motivation in their model [47, 55, 67, 68, 135]. On the other hand, studies on workers' motivation [25, 33, 79, 108, 114] did not investigate task assignment. Because we aim at studying two settings — *individual* and *holistic* task assignment, we need to design two models. To the best of our knowledge, no previous study has combined *adaptive* task assignment and *motivation-aware* task assignment. We need to devise a model that combine these two ideas.

Our third challenge is to design efficient algorithms for task assignment. Previous studies [106, 109] included the design of efficient approximation algorithms or leveraged indices to speed up task assignment. In an *online* context, designing a responsive system is crucial. Moreover, a worker should not remain idle while waiting for being assigned new tasks. This motivates the need to study efficient assignment algorithms.

### 3.1.2 Overview of our Contributions

This chapter presents our investigations on motivation-aware task assignment. Our investigations are organized as follows:

(a) We formalize a general data model and study motivation factors in Section 3.2. As a first attempt, we model three motivation factors: *task diversity*, *task payment* and *task relevance.*

(b) We study *Individual Task Assignment* (Ita) in Section 3.3 [101]. Here, we consider optimizing task assignment for each worker, *individually.*

    (a) We define motivation as a combination of two motivation factors: *task diversity* and *task payment.* We model the Ita problem and show it is NP-Hard.

    (b) We design and compare three task assignment strategies: (1) RELEVANCE, a strategy that chooses tasks that *match* a worker's profile, (2) DIVERSITY, a strategy that chooses *matching and diverse* tasks, and (3) DIV-PAY, an *adaptive* strategy that selects *matching* tasks with *the best compromise between diversity and payment.*

    (c) We conduct experiments with real workers hired from Amazon Mechanical Turk (AMT) to evaluate these strategies. Our empirical validation shows that different strategies prevail for different dimensions. RELEVANCE outperforms both DIV-PAY and DIVERSITY on task throughput and worker retention. However, DIV-PAY outperforms the other strategies on outcome quality, which confirms the need for an adaptive motivation-aware approach in crowdsourcing.

(c) We study *Holistic Task Assignment* (Hta) in Section 3.4 [102]. Here, we consider assigning tasks *holistically*, to a set of workers.

   (a) We define motivation as a combination of two factors: *task diversity* and *task relevance*. We model Hta and show it is NP-Hard and also Max-SNP-Hard.

   (b) We propose two approximation algorithms for Hta: HTA-APP and HTA-GRE. Both algorithms run in polynomial time and have $1/4^{th}$ and $1/8^{th}$ approximation factors, respectively. HTA-APP has a better approximation factor with the cost of a higher running time. While HTA-APP uses the Hungarian Algorithm as a sub-routine to find an *optimal* solution for an *auxiliary* assignment problem, HTA-GRE uses an *approximation algorithm* that has a better running time.

   (c) We run a simulation with synthetic workers, to examine scalability and the value of the objective function. We show that HTA-GRE outperforms HTA-APP since it has a better running time and returns solutions with a similar performance with regard to the objective function.

   (d) We conduct live experiments with real workers to study the end-to-end performance of our system. We show that optimizing diversity results in the highest crowdwork quality and relevance only is the worst. We find that HTA-GRE, offers the best compromise between performance dimensions thereby assessing the need for motivation-aware task assignment.

## 3.2 Data Model and Motivation Factors

In this section, we first describe our model for tasks and workers. Then, we present how we model the adaptive task assigment process and we present three motivation factors. Our model is applicable in our two variants of motivation-aware task assignment. Table 3.1 summarizes important notations used throughout this chapter.

Our investigation includes the formalization of three kinds of functions for motivation factors. For each factor, we define the *expected* motivation induced by a set of tasks. Then, we define how to capture the importance of each motivation factor *after having observed a worker completing tasks.*

### 3.2.1 Data Model for Tasks and Workers

We consider a set of tasks $\mathcal{T} = \{t_1, \dots, t_N\}$, a set of workers $\mathcal{W} = \{w_1, \dots, w_Q\}$ and a set of keywords $\mathcal{S} = \{s_1, \dots, s_R\}$.

| Notation | Definition |
|---|---|
| $\mathcal{T}$ | A set of tasks $\{t_1, \dots, t_N\}$ |
| $\mathcal{W}$ | A set of workers $\{w_1, \dots, w_Q\}$ |
| $\mathcal{S}$ | A set of skill keywords $\{s_1, \dots, s_R\}$ |
| $\mathcal{T}^i$ | Tasks available at iteration $i$ |
| $\mathcal{W}^i$ | Workers available at iteration $i$ |
| $\mathcal{T}^i_w$ | Tasks assigned to worker $w$ at iteration $i$ |
| $d(t_k, t_l)$ | Pairwise task diversity between two tasks |
| $TD(\mathcal{T}')$ | Task diversity of a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ |
| $TP(\mathcal{T}')$ | Task payment of a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ |
| $TR(\mathcal{T}', w)$ | Task relevance of a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ for the worker $w$ |
| $motiv^i_w(\mathcal{T}^i_w)$ | Expected motivation of worker $w$ on tasks $\mathcal{T}^i_w$ |
| $X_{max}$ | Maximum number of tasks assigned to a worker |

Table 3.1: Motivation-aware task assignment: summary of important notations

**Tasks.** A task $t$ is represented by a vector $\langle t(s_1), t(s_2), \dots, t(s_R), c_t \rangle$ where each $t(s_i)$ is a Boolean value that denotes the presence or absence of keyword $s_i$ in task $t$. A keyword associated to a task reflects its content and requirements. In Amazon Mechanical Turk, for instance, an audio transcription task is often associated with keywords such as *"audio"*, *"English"*, and *"news"*, while a video tagging task is associated with keywords such as *"Google street view"* and *"tagging"*. In Crowdflower, a sentiment analysis task is often associated to *"sentiment analysis"* and *"English."*. The reward $c_t$ is given to a worker who completes $t$.

**Workers.** A worker $w$ is a vector $w = \langle w(s_1), \dots, w(s_R) \rangle$ where each $w(s_i)$ is a Boolean value capturing the *expressed interest* of $w$ in tasks with keyword $s_i$.

*Example* (Tasks and Workers). Table 3.2 shows an example with 3 tasks, 2 workers and 5 skills. For instance, $t_1$ is characterized by a vector $\langle \texttt{true}, \texttt{true}, \texttt{false}, \texttt{false}, \texttt{false}, 0.01 \rangle$: it is an audio transcription task with a \$0.01 reward, and it is described by skill keywords "*audio*" and "*English*". $w_1$ is a worker who expresses interest in tasks that feature the keywords "*audio*" and "*tagging*". We could suppose that only workers covering all task skills are qualified to complete a task. In this example, $w_1$ would only qualify for task $t_2$, while $w_2$ would qualify for both $t_1$ and $t_3$.

| | *audio* | *English* | *French* | *review* | *tagging* | *reward* ($) |
|---|---|---|---|---|---|---|
| $t_1$ | ✓ | ✓ | | | | 0.01 |
| $t_2$ | | | | | ✓ | 0.03 |
| $t_3$ | | | ✓ | ✓ | | 0.09 |
| $w_1$ | ✓ | | | | ✓ | N/A |
| $w_2$ | ✓ | ✓ | ✓ | ✓ | | N/A |

Table 3.2: Example of tasks and workers

### 3.2.2 Adaptive Task Assignment Model

We advocate a multi-step approach where the set of tasks assigned to a worker are revisited at each step in order to best fit the worker's motivation. Figure 3.2 illustrates our adaptive process. At each iteration $i$ a set of tasks $\mathcal{T}_w^i \subseteq \mathcal{T}^i$ is assigned to a worker $w$. Here, $\mathcal{T}^i \subseteq \mathcal{T}$ refers to the set of available tasks at iteration $i$. We aim to assign a set of tasks that best match a worker's motivation. Therefore we need to capture her motivation based on the tasks that she completed during the previous iteration, when she was assigned tasks $\mathcal{T}_w^{i-1}$.

### 3.2.3 Motivation Factors

Several factors could be used to reflect workers' motivation. While some workers look for fun and enjoyment, others want to learn something new, pass time, or make some money. The six factors that influence motivation the most are *Payment, Task Autonomy, Task Diversity (a.k.a. Skill Variety), Task Identity, Human Capital Advancement, Pastime* [79]. In addition, since workers self-assign tasks to themselves, the relevance of a task to a worker is always implicitly present.

In this thesis, we propose to define motivation *as a balance* between several factors. As a first attempt, we propose to consider (i) *task diversity*, i.e., how different tasks are from each other, (ii) *payment*, that characterizes how well a set of tasks pays, and (iii) *task relevance*, i.e., how proficient a worker believes to be for a task. We aim to study two of these factors in the two variants of motivation-aware task assignment.

These factors are easily measurable and can be updated on-the-fly as workers complete tasks. Compared to other dimensions, only these dimensions are most relevant in micro-tasks and in typical labor markets such as Amazon Mechanical Turk. Task autonomy and task identity are often minimal since micro-tasks do not yield a high degree of freedom and usually do not allow workers to have a tangible result of their work. Human capital advancement, i.e., choosing a task for its capacity to increase one's knowledge, is also negligible. Finally, picking a task to pass time is difficult to measure.

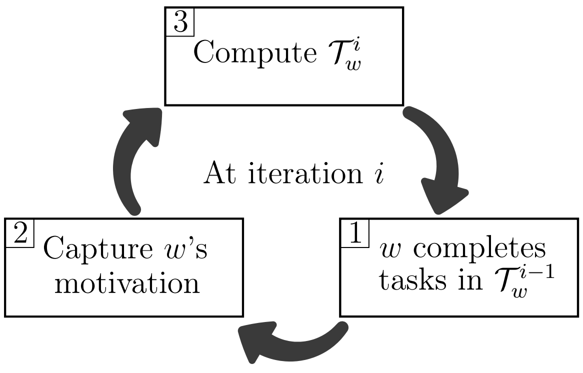

3 Compute $\mathcal{T}_w^i$
At iteration $i$
2 Capture $w$'s motivation
1 $w$ completes tasks in $\mathcal{T}_w^{i-1}$

Figure 3.2: Motivation-aware task assignment: model

Our formalization aims to quantify how diverse a set of task is (for the task diversity factor), pays well (task payment) or how relevant it is to a given worker (task relevance). Therefore, we aim to define a function for each factor, that takes a set of tasks as input (and if necessary a worker) and returns a value indicating how well the given factor is expected to motivate a worker.

**Task Diversity.** We first define $d(t_k, t_l)$, the *pairwise task diversity* between two tasks $t_k$ and $t_l$. It essentially measures the aggregated differences of keywords between two tasks. Task diversity has been qualified as a fun and enjoyment motivation factor [62]. In our setting, we use the Jaccard similarity function $J$ as follows:

$$d(t_k, t_l) = 1 - J(\langle t_k(s_1), \dots, t_k(s_R)\rangle, \langle t_l(s_1), \dots, t_l(s_R)\rangle)$$

In practice, we allow $d()$ to be any distance function (e.g., Euclidean, Jaccard) as long as it is a metric, i.e., it verifies the triangle inequality. The diversity $TD(\mathcal{T}')$ of a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ is captured in the usual manner, by aggregating the pairwise distances in $\mathcal{T}'$:

$$TD(\mathcal{T}') = \sum_{\substack{(t_k,t_l)\in\mathcal{T}' \\ k>l}} d(t_k, t_l) \tag{3.1}$$

**Task Payment.** The total task payment of a set of tasks $\mathcal{T}' \subseteq \mathcal{T}$ is the sum of individual task payments in $\mathcal{T}'$:

$$TP(\mathcal{T}') = \frac{1}{\max_{t\in\mathcal{T}} c_t} \times \sum_{t\in\mathcal{T}'} c_t \tag{3.2}$$

The denominator $\max_{t\in\mathcal{T}'} c_t$ normalizes each member of the sum in the interval $[0, 1]$.

**Task Relevance.** Let $rel(t, w)$ be the function that evaluates how relevant a task $t$ is to a worker $w$: We set:

$$rel(t, w) = 1 - d_{rel}(\langle t_{s_1}, \dots, t_{s_R}\rangle, \langle w_{s_1}, \dots, w_{s_R}\rangle)$$

We also use Jaccard to express $d_{rel}()$. The relevance $TR(\mathcal{T}', w)$ of a set of tasks $\mathcal{T}'$ for a worker $w$ is captured by aggregating each pairwise distance for $t \in \mathcal{T}'$.

$$TR(\mathcal{T}', w) = \sum_{t \in \mathcal{T}'} rel(t, w) \tag{3.3}$$

### 3.2.4 Capturing Motivation

We have defined the *expected* motivation induced by a set of tasks for a given worker according to a specific motivation factor. We now need to model how we capture a worker's *actual* motivation at each iteration $i$. Our end goal is to enable task assignment by capturing workers' motivation and incorporating it in task assignment in the next iteration. Therefore, for each worker $w$ we aim to leverage completed tasks in $\mathcal{T}_w^{i-1}$, that were assigned during iteration $(i-1)$.

We propose to quantify how motivated a a worker is by a given factor *each time she completes a task.* Our rationale is as follows. Each time $w$ completes a task $t_j \in \mathcal{T}_w^{i-1}$, we can learn about her motivations: she may have preferred a task that differs from the tasks previously completed, a task that pays high or a task that best matches her interests. She may have also chosen a task that matches all these characteristics. We aim to leverage a collection of such observations to capture her motivation.

Suppose that when worker $w$ completes task $t_j \in \mathcal{T}_w^{i-1}$, she has already completed tasks $\{t_1, \ldots, t_{j-1}\}$ where $j-1 \in 1, \ldots, |\mathcal{T}_w^{i-1}| - 1$. We capture each factor as follows.

**Task Diversity.** Equation 3.4 shows how we capture the gain in task diversity that a worker $w$ seeks when picking a task $t_j$ in the remaining tasks $\mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\}$.

$$\Delta TD(t_j) = \frac{\sum_{k \in 1,\ldots,j-1} d(t_j, t_k)}{\max_{t_{k'} \in \mathcal{T}_w^{i-1} \setminus \{t_1,\ldots,t_{j-1}\}} \sum_{t_k \in \{t_1,\ldots,t_{j-1}\}} d(t_{k'}, t_k)} \tag{3.4}$$

The numerator is the marginal gain in diversity when $w$ selects a task $t_j$. The denominator reflects the maximum possible marginal gain when $w$ selects a task in the remaining tasks $\mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\}$. When the denominator returns 0, we set $\Delta TD(t_j) = 1$.

**Task Payment.** We compute the list of all *different* task payments in $\mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\}$ and sort it by descending order. Suppose that this list counts $R$ elements and that $r(t_j)$ is the rank of $c_{t_j}$ in this list (if $c_{t_j}$ is the highest then $r(t_j) = 1$). We define $TP\text{-}Rank(t_j) \in [0, 1]$ such that $TP\text{-}Rank(t_j) = 1$ *iff* $t_j$ has the highest payment (0 if it has the lowest payment):

$$TP\text{-}Rank(t_j) = 1 - \frac{r(c_{t_j}) - 1}{R - 1} \tag{3.5}$$

Equation 3.5 captures the willingness of $w$ to choose tasks that pay highly among the available tasks. When the denominator returns 0, we set $TP\text{-}Rank(t_j) = 1$.

*Example* 2. Suppose that $\mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\} = \{t_5, t_6, t_7, t_8\}$ with $c_{t_5} = \$0.03, c_{t_6} = c_{t_7} = \$0.02, c_{t_8} = \$0.04$. A worker $w$ selects $t_5$, which has the second highest task payment among the remaining tasks. We obtain *TP-Rank*$(t_5) = 1 - \frac{2-1}{3-1} = 0.5$.

**Task Relevance.** We adopt the same rationale as in task diversity. Equation 3.6 shows how we capture the gain in task relevance that a worker $w$ seeks when picking a task $t_j$ in the remaining tasks $\mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\}$.

$$\Delta TR(t_j, w) = \frac{rel(t_j, w)}{\max\limits_{t_k \in \mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\}} rel(t_k, w)} \tag{3.6}$$

The numerator is the marginal gain in relevance when $w$ selects a task $t_j$. The denominator reflects the maximum possible marginal gain when $w$ selects a task in the remaining tasks $\mathcal{T}_w^{i-1} \setminus \{t_1, \ldots, t_{j-1}\}$. When the denominator returns 0, we set $\Delta TR(t_j, w) = 1$.

## 3.3 Individual Motivation-Aware Task Assignment

In this section, we present a first variant of Motivation-Aware Task Assignment: Individual Task Assignment (Ita) [101]. First, we formalize the Ita problem and show that it is NP-Hard. Then we present assignment strategies to solve Ita and we conduct live experiments with real workers to evaluate these strategies.

### 3.3.1 Individual Task Assignment Problem (Ita)

We aim to assign tasks to workers, *individually*. Therefore, each worker is related to an independent series of assignment iterations. Equivalently, an assignment iteration is related to a *single* worker. Consequently, we would need to revisit our model and consider that each worker $w$ is assigned a set of tasks $\mathcal{T}_w^{i_w}$ at each iteration $i_w$. For the sake of simplicity, we do not use this notation and denote as $\mathcal{T}_w^i$ the set of tasks assigned to worker $w$ at iteration $i$.

We first define the expected motivation of worker $w$ on tasks $\mathcal{T}_w^i$. In Ita, we propose to focus on *task diversity* and *task payment*. We define the function $motiv_w^i$ as a linear combination of diversity and payment of tasks in $\mathcal{T}_w^i$:

$$motiv_w^i(\mathcal{T}_w^i) = 2\alpha_w^i \times TD(\mathcal{T}_w^i) + (|\mathcal{T}_w^i| - 1)(1 - \alpha_w^i) \times TP(\mathcal{T}_w^i) \tag{3.7}$$

$\alpha_w^i \in [0, 1]$ is a worker-specific parameter that reflects the relative importance between *task diversity* and *task payment*. It represents the compromise a worker $w$ is looking for in choosing tasks to complete at each iteration $i$. This parameter hence captures the motivation of worker $w$ at iteration $i$.

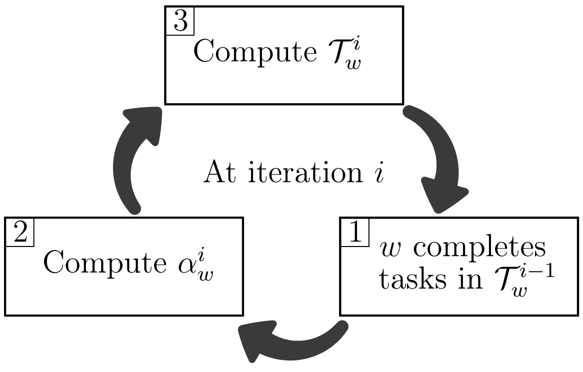

3
Compute $\mathcal{T}_w^i$
At iteration $i$
2
Compute $\alpha_w^i$
1
$w$ completes tasks in $\mathcal{T}_w^{i-1}$

Figure 3.3: Task assignment in Ita

We normalize the two components of the function with the factors $(|\mathcal{T}_w^i|-1)$ and 2, since the first part of the sum counts $\frac{|\mathcal{T}_w^i|(|\mathcal{T}_w^i|-1)}{2}$ numbers and the second part $|\mathcal{T}_w^i|$ numbers [59].

Note that we define the function as a linear formula between diversity and payment of a task, instead of a more complex non-linear formula, since a linear formula is likely to give rise to algorithms with theoretical guarantees as we show later, and is easier to interpret and explain.

*Example* (Worker's compromise in Ita). A worker $w_1$ with $\alpha_w^i = 0.1$ would be interested more in high-paying tasks with similar keywords (i.e., less diversity). This worker $w_1$ would choose tasks with a variety of keywords only if the payment is high enough. On the other hand, a worker $w_2$ with $\alpha_w^i = 0.9$ would be more motivated by task diversity.

Now, we formally define the individual motivation-aware ask assignment problem, Ita, as follows:

**Problem 1** (Individual Motivation-Aware Task Assignment — Ita)**.** For any worker $w \in \mathcal{W}$, at each iteration $i$, choose a set of tasks $\mathcal{T}_w^i \subseteq \mathcal{T}^i$ such that:

$$\begin{aligned} &\textbf{max} && motiv_w^i(\mathcal{T}_w^i) && \\ &\textit{such that} && \forall t \in \mathcal{T}_w^i \; matches(w,t) && (C_1) \\ & && |\mathcal{T}_w^i| \leq X_{max} && (C_2) \end{aligned}$$

The function $matches(w,t)$ in constraint $C_1$ returns `true` if the task $t$ *matches* worker $w$. We can use various definitions for $matches(w,t)$. For instance, we can define $matches(w,t) =$ `true` iff the skill keywords of $w$ and $t$ are identical. In our setting, we suppose that $matches(w,t)$ captures how well the skill keywords of $w$ *cover* the skill keywords of $t$. This allows us to capture cases where $w$ *matches* $t$ only if $w$ expresses interest in at least 50% of the skill keywords of $t$. $X_{max}$ is used in constraint $C_2$ to avoid assigning too many tasks to workers with varied interests, and reflects the ability of a worker to explore only a few tasks at a time (akin to limiting Web search results).

We suppose that each time we solve the Ita problem for a given worker $w$, $w$ matches at least $X_{max}$ tasks. Thus, given that the objective function is positive and monotonically increasing, $w$ will be assigned *exactly* $X_{max}$ tasks. That is a realistic assumption when $X_{max}$ is chosen to be reasonably small (e.g., 20) in a context where we have a large collection of tasks to assign.

The Ita problem considers each worker *individually*. When a worker $w$ requires a new set of tasks $\mathcal{T}_w^i$, Ita is solved and tasks in $\mathcal{T}_w^i$ are dropped from $\mathcal{T}^i$. Thus, a task is assigned to at most one worker. Figure 3.3 illustrates our process in Ita.

#### 3.3.1.1 NP-Hardness of Ita

Intuitively, the Ita problem is difficult since it aims at finding a set that maximizes a sum of pairwise distances, a common feature in several well-known NP-hard problems. In particular, Ita is closely related to the maximum dispersion problem (MaxSumDisp) [26, 49, 65, 107].

**Theorem 1.** *The motivation-aware task assignment problem (Ita) is NP-hard.*

*Proof.* At each iteration and for each worker, the decision version of Ita is as follows.

*Instance*: Tasks $\mathcal{T}^i$, worker $w$ and her $\alpha_w^i$, $X_{max}$ and an objective value $Z$. *Question*: Is there a set $\mathcal{T}_w^i \subseteq \mathcal{T}$ such that $C_1$ is satisfied, $|\mathcal{T}_w^i| = X_{max}$ and $motiv_w^i(\mathcal{T}_w^i) \geq Z$ ? Ita $\in$ *NP* since a non-deterministic algorithm needs only to guess a set $\mathcal{T}_w$ and it can verify the question in polynomial time.

*Reduction of Max Dispersion.* To prove the NP-hardness, we consider the maximum sum dispersion problem (MaxSumDisp) [26, 49, 65, 107] (also known as *Maximum Edge Subgraph*)[1]. The decision version of this problem is as follows.

*Instance*: a complete weighted graph $G = (V, E, \omega)$, an integer $k \in [2, |V|]$, an objective value $Y$. *Question*: Is there $V' \subseteq V$ such that $|V'| = k$ and $\sum_{\{v_1,v_2\} \in E \cap \{V' \times V'\}} \omega(v_1, v_2) \geq Y$?

Note that MaxSumDisp is well-known to be NP-hard[2] [18, 26, 49, 107] using a reduction from MaxClique [56]. Because a non-deterministic algorithm can guess a solution $V'$ and easily verify it in polynomial time, MaxSumDisp $\in$ *NP*. Thus, MaxSumDisp is also NP-Complete. The reduction works as follows: each vertex $v \in V$ is mapped to a task $t_v \in \mathcal{T}$. The weight of an edge $\{v_1, v_2\} \in E$ is mapped to skill variety between two tasks: $\omega(v_1, v_2) = 2 * d(t_{v_1}, t_{v_2})$. We consider that $\alpha_w^i = 1$. We set $X_{max} = k$ and $Z = Y$. This creates an instance of Ita in polynomial time. This instance has the objective function $2 * \sum_{t_k, t_l \in \mathcal{T}_w^i\ k>l} d(t_k, t_l)$. MaxSumDisp has a solution if and only if this instance of Ita has a solution. This proves the NP-hardness. □

[1] http://www.nada.kth.se/~viggo/wwwcompendium/node46.html

[2] MaxSumDisp is NP-Hard even if the weights satisfy the triangle inequality [107].

### 3.3.2 Our Approach for Ita

In this section, we present the two main components of motivation-aware task assignemnt. First, we present how we capture the expected motivation of a worker. For a given worker $w$, we show how we define $\alpha_w^i$ at each iteration. Second, in order to study the effect of different dimensions in our problem, we explore approaches that exploit different objectives in the Ita problem. First, we design RELEVANCE, a diversity and payment-agnostic strategy. This strategy focuses on assigning to workers tasks that best match their interests. Second, we present DIV-PAY, that optimizes the objective function of the Ita problem. DIV-PAY is hence both diversity and payment-aware. Third, we present DIVERSITY, that focuses only on assigning diverse tasks to workers and is hence payment-agnostic.

#### 3.3.2.1 Computing $\alpha_w^i$

In Section 3.2, we defined how to capture the importance of each factor (Equations 3.4 and 3.5) when a worker chooses a task in the set of available tasks. We now need to define $\alpha_w^{ij}$, that captures the compromise between task diversity and task payment that worker $w$ seeks when selecting task $t_j \in \mathcal{T}_w^{i-1}$. We set:

$$\alpha_w^{ij} = \frac{\Delta \mathit{TD}(t_j) + 1 - \mathit{TP\text{-}Rank}(t_j)}{2} \tag{3.8}$$

$\alpha_w^{ij}$ is defined as the average of $\Delta \mathit{TD}(t_j)$ and $1 - \mathit{TP\text{-}Rank}(t_j)$. The asymmetry comes from the fact that the higher $\alpha_w^i$ is, the lower is the importance of the task payment factor. We can observe that if both $\Delta \mathit{TD}(t_j)$ and $\mathit{TP\text{-}Rank}(t_j)$ return the same value, $\alpha_w^{ij}$ will be equal to 0.5.

Having defined each $\alpha_w^{ij}$, we are now ready to capture $\alpha_w^i$. Suppose that during iteration $i-1$ worker $w$ chose $J$ tasks where $J \leq |\mathcal{T}_w^{i-1}|$. We compute $\alpha_w^i$ as the average of all $\alpha_w^{ij}$ for $j \in 2, \ldots, J$:

$$\alpha_w^i = \operatorname*{avg}_{j \in 2,\ldots,J} \alpha_w^{ij} \tag{3.9}$$

We skip the case where $j = 1$ that corresponds to the first completed task, since the marginal gain in task diversity would return 0 (Equation 3.4).

#### 3.3.2.2 Relevance strategy

We first propose the RELEVANCE approach (Algorithm 5), that assigns $X_{max}$ random tasks that match workers' interests. RELEVANCE enforces constraints $C_1$ and $C_2$ and ignores task diversity and task payment. At each iteration $i$ and for each worker $w$, RELEVANCE (i) filters the tasks $\mathcal{T}_{match(w)}$ that match $w$ and (ii) selects randomly $X_{max}$ tasks among $\mathcal{T}_{match(w)}$. In this strategy, a worker's motivation is therefore interpreted as matching her interests.

**Algorithm 5** RELEVANCE

```
Input: $\mathcal{T}^i$, $w, X_{max}, i$
Output: $\mathcal{T}_w^i$
1: $\mathcal{T}_w^i \leftarrow \emptyset$
2: $\mathcal{T}_{match(w)} \leftarrow \{t \in \mathcal{T}^i \mid matches(w,t)\}$
3: while $|\mathcal{T}_w^i| < X_{max}$
4:     $\mathcal{T}_w^i \leftarrow \mathcal{T}_w^i \cup \{\text{nextRandomTask}(\mathcal{T}_{match(w)} \setminus \mathcal{T}_w^i)\}$
   return $\mathcal{T}_w^i$
```

**Algorithm 6** DIV-PAY

```
Input: $\mathcal{T}^i$, $w, X_{max}, i, \mathcal{T}_w^{i-1}, \{t_1, \ldots, t_J\}$ tasks completed in iteration $i-1$
Output: $\mathcal{T}_w^i$
1: $\alpha_w^i \leftarrow \text{avg}_{k \in [\![2,J]\!]}\, \alpha_w^{ij}$
2: $\mathcal{T}_{match(w)} \leftarrow \{t \in \mathcal{T}^i \mid matches(w,t)\}$
3: $\mathcal{T}_w^i \leftarrow$ GREEDY$(\mathcal{T}_{match(w)}, X_{max}, w)$
4: return $\mathcal{T}_w^i$
```

#### 3.3.2.3 Diversity and payment-aware strategy

We present DIV-PAY, a strategy that is both diversity and payment-aware. DIV-PAY relies on both the on-the-fly estimation of a worker's motivation, and the online iterative task assignment. The motivation of a worker $w$ is captured in the value of $\alpha_w^i$, which represents the *compromise* a worker $w$ is looking for in choosing tasks to complete at each iteration $i$.

**Assigning Tasks.** At each iteration $i$, the DIV-PAY strategy aims to solve the Ita problem for each worker. We now present the DIV-PAY algorithm that returns a solution with an approximation ratio of 2 for the Ita problem.

**Algorithm.** Because it is an NP-hard problem, Ita is prohibitively expensive to solve on large instances. In our scenario, response time is important since Ita has to be solved *online*, at each iteration $i$. The good news is that approximation algorithms exist for some related problems, such as MaxSumDisp [64, 65, 107] if the distance $d$ satisfies the triangle inequality. The assumption that the distance obeys triangle inequality is not an overstretch, as many real world distances satisfy this assumption [16]. We note that the pairwise task diversity defined in Section 3.2 is a metric and follows triangle inequality.

We adapt an existing algorithm for the maximum diversification problem MaxSumDiv (which includes MaxSumDisp as a special case). We design DIV-PAY (Algorithm 6) that assigns a set of tasks $\mathcal{T}_w^i$ to a worker $w$. First, DIV-PAY captures the motivation of a worker $w$ in the value of $\alpha_w^i$. Then DIV-PAY computes a set of matching tasks (line 2) and runs

**Algorithm 7** GREEDY [18]

```
Input: $\mathcal{T}_{match(w)}, X_{max}, w, i$
Output: $\mathcal{T}_w^i$
1: $\mathcal{T}_w^i \leftarrow \emptyset$
2: while $|\mathcal{T}_w^i| < X_{max}$
3:     $t \leftarrow \underset{t' \in \mathcal{T}_{match(w)} \setminus \mathcal{T}_w^i}{\text{argmax}} g(\mathcal{T}_w^i, t')$
4:     $\mathcal{T}_w^i \leftarrow \mathcal{T}_w^i \cup \{t\}$
   return $\mathcal{T}_w^i$
```

GREEDY that returns $\mathcal{T}_w^i$ (line 3). GREEDY (Algorithm 7) is a $\frac{1}{2}$-approximation algorithm for the MaxSumDiv problem [18]. In the MaxSumDiv problem, the objective is to find a set $S$ of $p$ elements that maximizes

$$\lambda \sum_{\{u,v\}:u,v \in S} d(u,v) + f(S)$$

$\lambda$ is a weight parameter, $f(S)$ is a normalized, monotone submodular function measuring the value of $S$ and $d()$ is a distance function evaluating the diversity between two elements. Since Ita simplifies to finding a set of size *exactly* $X_{max}$, the objective function can be rewritten as:

$$motiv_w^i(\mathcal{T}_w^i) = 2\alpha_w^i \times TD(\mathcal{T}_w^i) + (1 - \alpha_w^i)(X_{max} - 1) \times TP(\mathcal{T}_w^i)$$

Now, we map our problem to the MaxSumDiv problem by setting $f(\mathcal{T}_w^i) = (X_{max} - 1) \times (1 - \alpha_w^i) \times TP(\mathcal{T}_w^i)$, $\lambda = 2\alpha_w^i$ and $p = X_{max}$. It can be easily seen that $f$ is normalized ($f(\emptyset) = 0$). $f$ is monotone since $\forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{T}$ s.t. $\mathcal{T}_1 \subseteq \mathcal{T}_2$ we have $f(\mathcal{T}_1) \leq f(\mathcal{T}_2)$. It is also submodular since $\forall \mathcal{T}_1, \mathcal{T}_2 \subseteq \mathcal{T}$ s.t. $\mathcal{T}_1 \subseteq \mathcal{T}_2$ and $\forall t \in \mathcal{T}$, we have:

$$\begin{aligned} f(\mathcal{T}_1 \cup \{t\}) - f(\mathcal{T}_1) &= (X_{max} - 1)(1 - \alpha_w^i) \times \frac{1}{\max_{t' \in \mathcal{T}} c_{t'}} \times c_t \\ &= f(\mathcal{T}_2 \cup \{t\}) - f(\mathcal{T}_2) \end{aligned}$$

At each iteration, GREEDY inserts in $\mathcal{T}_w^i$ the task $t$ that maximizes the function $g(\mathcal{T}_w^i, t) = \frac{1}{2}(f(\mathcal{T}_w^i \cup \{t\}) - f(\mathcal{T}_w^i)) + \lambda \sum_{t' \in \mathcal{T}_w^i} d(t,t')$ which is equal to $g(\mathcal{T}_w^i, t) = (X_{max} - 1)(1 - \alpha_w^i) TP(\{t\})/2 + 2\alpha_w^i \sum_{t' \in \mathcal{T}_w^i} d(t,t')$.

We run GREEDY using tasks that verify the constraint $C_1$ (Algorithm 6, line 2), thus the algorithm returns a correct solution for the Ita problem. Because GREEDY is a $\frac{1}{2}$-approximation algorithm for the MaxSumDiv problem, DIV-PAY is a $\frac{1}{2}$-approximation for the Ita problem. Borodin et al. [18] observe that the GREEDY algorithm runs in time linear in the number of input elements when the desired size of the set is a constant. In our setting, we can conclude that DIV-PAY runs in $\mathcal{O}(X_{max} * |\mathcal{T}|)$ time.

**Algorithm 8** DIVERSITY

```
Input: $\mathcal{T}^i, w, X_{max}, i$
Output: $\mathcal{T}^i_w$
1: $\alpha^i_w \leftarrow 1$
2: $\mathcal{T}_{match(w)} \leftarrow \{t \in \mathcal{T}^i \mid matches(w,t)\}$
3: $\mathcal{T}^i_w \leftarrow$ GREEDY$(\mathcal{T}_{match(w)}, X_{max}, w)$
4: return $\mathcal{T}^i_w$
```

One may wish to extend the motivation model used in Ita. We observe that the performance guarantee and the running time of GREEDY hold as long as our objective function has the form $\lambda \sum_{\{u,v\}:u,v \in S} d(u,v) + f(S)$ where $f$ is a normalized, monotone and submodular function.

#### 3.3.2.4 Diversity strategy

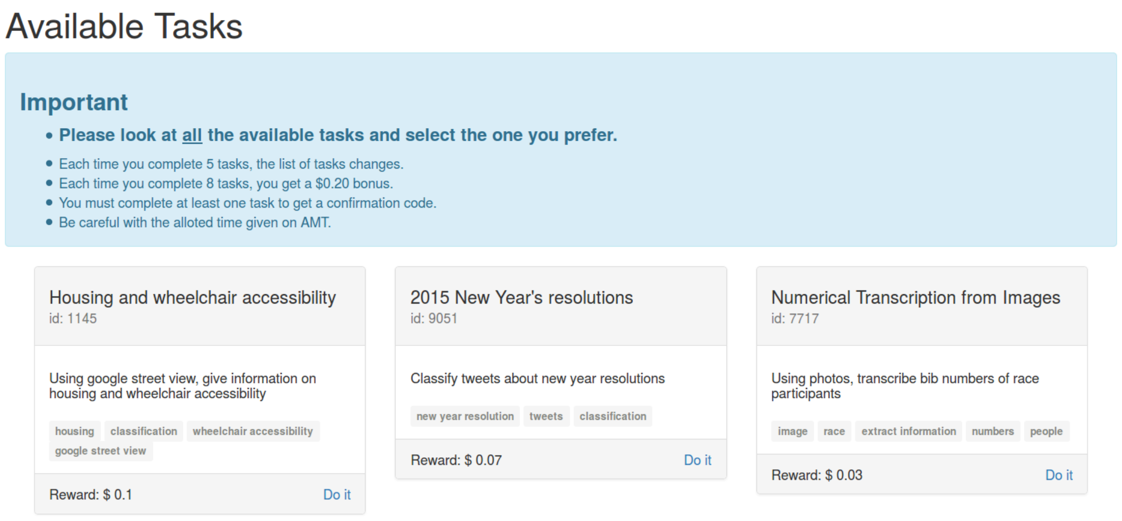
We propose DIVERSITY (Algorithm 8), a strategy that is diversity-aware and payment-agnostic. DIVERSITY considers a variant of the Ita problem where the objective includes only the task diversity sum. DIVERSITY employs GREEDY as a subroutine with $\alpha^i_w = 1$ at every iteration. We can follow the same reasoning exposed for Ita: constraint $C_1$ is enforced on line 2 and GREEDY is a $\frac{1}{2}$-approximation for the considered problem, so DIVERSITY is a $\frac{1}{2}$-approximation for this variant of the Ita problem.

### 3.3.3 Experiments

#### 3.3.3.1 Workflow

To achieve adaptability, we developed a new platform, GACS (Grenoble Adaptive Crowd-Sourcing). During a work session, a (i) worker chooses keywords to build her keyword vector, (ii) completes tasks and (iii) is (re)assigned tasks using one the three strategies RELEVANCE, DIV-PAY, DIVERSITY when a new assignment iteration is performed. An iteration happens only if a worker has completed 5 tasks. Appendix A.2 presents the workflow in details.

**Assignment Iterations.** For the strategies RELEVANCE and DIVERSITY, we run the according algorithm at each iteration. For the strategy DIV-PAY, we need to consider the first iteration ($i = 1$) where $w$ arrives for the first time on our platform. In this case, we cannot compute her $\alpha^1_w$ since she has not yet completed tasks. We hence use RELEVANCE as a cold-start assignment strategy in the first iteration. We aim to learn $w$'s preference between diversity and payment using a strategy that does not favor any factor. Our rationale is to get an accurate estimation of $\alpha^1_w$. At the next iterations, since $w$ has



Available Tasks

Important

- Please look at all the available tasks and select the one you prefer.
- Each time you complete 5 tasks, the list of tasks changes.
- Each time you complete 8 tasks, you get a $0.20 bonus.
- You must complete at least one task to get a confirmation code.
- Be careful with the alloted time given on AMT.

Housing and wheelchair accessibility
id: 1145
Using google street view, give information on housing and wheelchair accessibility
housing classification wheelchair accessibility google street view
Reward: $ 0.1 Do it

2015 New Year's resolutions
id: 9051
Classify tweets about new year resolutions
new year resolution tweets classification
Reward: $ 0.07 Do it

Numerical Transcription from Images
id: 7717
Using photos, transcribe bib numbers of race participants
image race extract information numbers people
Reward: $ 0.03 Do it

Figure 3.4: Example screenshot of user interface – e.g., task grid

already completed tasks, we run DIV-PAY. We compute her $\alpha_w^i$ and return the new set of tasks $\mathcal{T}_w^i$.

#### 3.3.3.2 Settings

**Tasks.** We used a set of $158,018$ micro-tasks released by Crowdflower. It includes 22 different kinds of tasks. Appendix A.1 presents this dataset. We ran experiments using a sample of $5,000$ tasks.

**Task assignment.** We conducted experiments with all task assignment strategies, RELEVANCE, DIV-PAY, and DIVERSITY. We adapted the RELEVANCE strategy because the distribution of tasks is not uniform in our dataset (some kinds of tasks are over represented). The random task selection was achieved by first selecting a random *kind* of task, and then selecting a random task of this particular kind. We set $X_{max} = 20$. We set $matches(w, t) = \texttt{true}$ *iff* $w$ is interested by at least 10% of the keywords of task $t$. Workers were asked to provide at least 6 keywords. We also verified the response time of our algorithms: any approach returned a solution in a few milliseconds upon a worker request. This makes our approaches suitable for an online setting: new workers and tasks can be easily handled by computing assignments from scratch.

**Workers and Payment.** We published 30 HITs on Amazon Mechanical Turk (AMT) to recruit workers. Each HIT invites a worker to complete tasks on our platforms. Our HITs were completed by 23 different workers. We assigned 10 HITs for each task assignment strategy. Appendix A.2 gives more details on workers recruitment. We set a HIT reward to \$0.1 and each worker was granted a bonus equivalent to the total reward of the tasks she completed. Additionally, we encouraged workers who completed many tasks: we granted them a \$0.2 bonus each time they completed 8 tasks. We required workers to have previously completed at least 200 HITs that were approved, and to have an approval rate above 80%. We also required HITs to be completed within 20 minutes: our rationale is to encourage workers to choose quickly tasks that they prefer.

**User Interface.** We conducted a first set of experiments where the $\mathcal{T}_w^i$ were displayed as a ranked list. We observed that most workers selected the top task first, completed it, and walked down the list in order. This created a bias and defeated our purpose: observing workers making choices based on their motivation. In order to reduce the effect of ranking, we changed the interface by showing a grid with 3 tasks per row (Figure 3.4). That mitigated the effect of ranking and workers stopped choosing the task in their order of appearance. We used the grid-based presentation in all our experiments.

**Evaluation measures.** We evaluate our task assignment strategies using three main performance measures. First, we evaluate outcome *quality* that reflects the quality of a worker's contribution to a task with regard to a ground truth. Then, we evaluate the *number of completed tasks* per session and per unit of time (i.e. *task throughput*). The higher the number of completed tasks and the higher the throughput, the faster a requester can have crowdwork completed. We also evaluate *worker retention* that characterizes the willingness of workers to work on our tasks. Additionally, we compare our strategies on task payment and we measure the worker-specific parameter $\alpha_w^i$ that captures the compromise worker $w$ is looking for at iteration $i$ between task diversity and task payment.

#### 3.3.3.3 Summary of Results

We summarize our results and provide a rationale for why different strategies prevail for different measures. We observe that RELEVANCE is the strategy that provides the best number of completed tasks and task throughput. This could be explained by the fact that RELEVANCE requires less effort from workers than DIVERSITY and DIV-PAY. Indeed, since RELEVANCE is based on selecting tasks that best match a worker's profile and since a worker's profile is quite homogeneous, tasks recommended by RELEVANCE are quite similar to each other. Therefore, a worker does not do much context switching between tasks and is hence faster overall. We also observe that DIV-PAY slightly outperforms DIVERSITY on number of completed tasks and task throughput. That shows the importance of task payment as an incentive. Results are different if we consider crowdwork quality. DIV-PAY is the strategy that obtains the best quality, followed by RELEVANCE. DIV-PAY is the only strategy that is both adaptive and motivation-aware: this contributes to providing a better incentive to workers. Quality comes at a price though: DIV-PAY is the strategy where the average task payment among completed tasks is the highest and it does not provide the highest task throughput (it is better than DIVERSITY but worse than RELEVANCE). Thus, depending on the platform, one should study trade-offs between these strategies when designing task assignment algorithms.

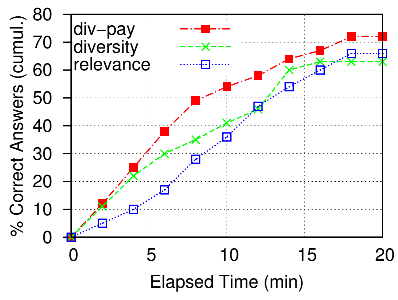
We observe that DIV-PAY rewarded workers higher. This could be expected since it is the only payment-aware strategy. Worker retention is better with RELEVANCE: workers performed longer work sessions with RELEVANCE than with other strategies. This finding is related to the fact that most workers do not have a clear preference for task diversity or task payment. They prefer tasks that match their interests and require fewer context switching, hence they did not necessarily stay longer when task diversity or task payment were favored (with DIV-PAY or DIVERSITY). We also observe workers' motivation and we notice that some workers carefully choose task diversity or task payment. In that case, we could accurately capture their preferences with appropriate $\alpha_w^i$ values. That allowed DIV-PAY to slightly outperform DIVERSITY on both task throughput and worker retention.

#### 3.3.3.4 Results

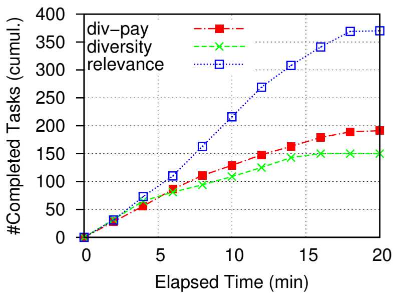
23 different workers completed 711 tasks in 30 work sessions. On average, each worker spent 13 minutes to submit a HIT and completed 23.7 tasks. On average, 73% of workers chose fewer than 10 keywords (6 is the minimum possible).

**Quality.** For each kind of task, we sampled 50% of completed tasks and we manually evaluated their ground truth. We chose tasks for which defining a ground truth was not controversial (e.g., a task that asks for the presence or not of a pattern on an image). Then, we compared each worker's contribution to a task to its ground truth. Figure 3.5a presents the cumulative percentage of tasks that were correctly completed for each strategy (see Table 3.5d for the total percentage). We observe that workers performed better with DIV-PAY (73% of correct answers) than with other strategies (DIVERSITY: 64%, RELEVANCE: 67%). The superiority of DIV-PAY over DIVERSITY is confirmed at the significance level 0.1 using two-proportion Z-test on the proportions of correct answers. This shows that assigning tasks that best match workers' compromise between task payment and task diversity encourages them to produce better answers. We observe that considering only task diversity (DIVERSITY) is not efficient. Including task payment is therefore important.

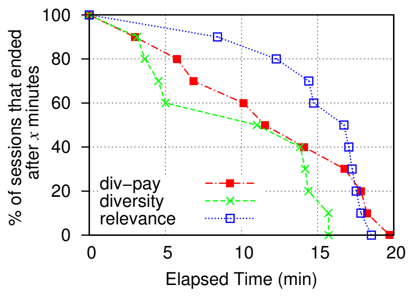
**Number of Completed Tasks.** Figure 3.5b presents the cumulative number of completed tasks (see Table 3.5d for the total). Overall, RELEVANCE clearly outperforms DIV-PAY, which is slightly better than DIVERSITY. Figure 3.5e details the number of completed tasks for each work session $h_k$, $k \in [\![1, 30]\!]$. We observe that with RELEVANCE, 5 sessions had more than 40 completed tasks. With DIV-PAY and DIVERSITY, most workers completed fewer than 30 tasks. The superiority of RELEVANCE over DIV-PAY and DIVERSITY is confirmed at the significance level 0.01 using Mann-Whitney U test on the number of completed task per session. We also measured the task throughput (i.e., number of completed tasks per minute). We considered the total time spent on our application, including



(a) Crowdwork quality



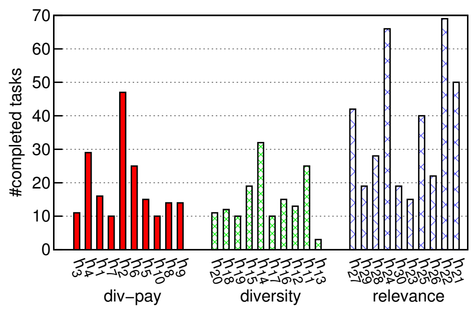
(b) Number of completed tasks



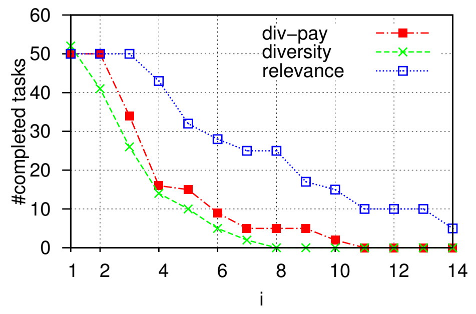
(c) Worker retention

| Strategy | % Correct answers | Avg. # completed tasks/session | Avg. session duration (min.) | # Completed tasks/min. | Total task payment ($) | Avg. task payment ($ cents) |
|---|---|---|---|---|---|---|
| DIV-PAY | 72.7% | 19.1 | 12.8 | 1.5 | 14.3 | 7.5 |
| DIVERSITY | 63.8% | 15.0 | 10.5 | 1.4 | 9.8 | 6.5 |
| RELEVANCE | 66.9% | 37.0 | 15.7 | 2.4 | 20.9 | 5.7 |

(d) Aggregated measures



(e) Number of completed tasks per work session



(f) Number of completed tasks per iteration

Figure 3.5: lta online experiments: results

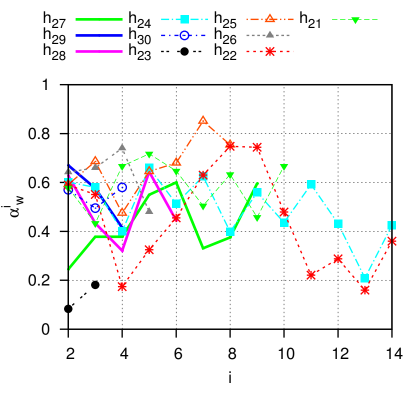
the time spent selecting a task to complete. The total time was higher with RELEVANCE (157 min) than with DIV-PAY (127 min). However, workers who were assigned tasks with RELEVANCE were more efficient (2.4 tasks/min. vs 1.5 tasks/min.). This could be explained by the fact that very little context switching is required from workers in the case of RELEVANCE (since tasks are both relevant to the worker and are potentially very similar to each other). DIVERSITY on the other hand, is slightly inferior to DIV-PAY. That leads us to the conclusion that workers did not necessarily appreciate diverse tasks, possibly for context-switching reasons. DIV-PAY slightly outperformed DIVERSITY, showing that including task payment as a motivating factor improves task throughput.

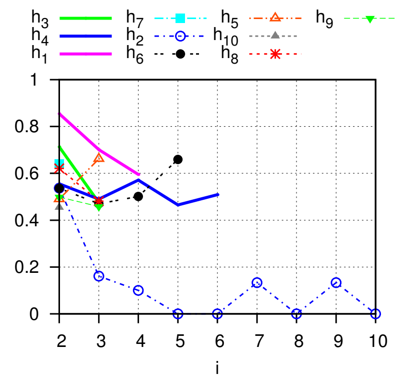
**Worker retention.** Figure 3.5c shows worker retention as the percentage of work sessions (vertical axis) that ended after $x$ minutes (horizontal axis). We find that workers stayed longer when they were assigned tasks using RELEVANCE, hence this approach clearly outperforms DIV-PAY and DIVERSITY. The average session duration using RELEVANCE was greater than with DIV-PAY and DIVERSITY (for the latter comparison the significance level is 0.1 using Mann-Whitney U test on session duration). Figure 3.5f supports this observation: more iterations were performed by workers with RELEVANCE. Although the number of completed tasks is roughly the same with all approaches on the first 2 iterations, this number fell quickly for DIV-PAY and DIVERSITY when $i > 2$. We also observe that DIV-PAY has a better worker retention than DIVERSITY. A plausible explanation is that workers are most comfortable completing similar tasks in a row. Therefore, they stay longer. They are least comfortable completing tasks with very different skills and tend to leave earlier. However, given that the quality of crowdwork reaches its best with DIV-PAY, we can say that optimizing for task relevance alone does not provide the best outcome quality even if workers are retained longer in the system.

**Task Payment.** We evaluate how rewarded workers were during their work sessions using our different assignment strategies. In crowdsourcing, both requesters and workers look for a fair treatment when it comes to compensation. Requesters look to obtain high-quality contributions at a reasonable rate, and workers expect to be adequately paid for their efforts. Table 3.5d presents the total task payment and the average payment per completed task for each strategy. The total payment is greater with RELEVANCE than with other approaches. This could be expected given the number of completed tasks (Section 3.3.3.4). However, the average task payment is the greatest with DIV-PAY. That is explained by the fact that DIV-PAY is the only strategy that is payment-aware. Thus, it is likely to assign higher-paying tasks to workers that prefer task payment over task diversity.

**Workers' motivation.** We now turn to workers and study their motivation in detail. In order to make a fair comparison, we compute $\alpha_w^i$ for each strategy and for each iteration $i \geq 2$ (even if it is only used by DIV-PAY). Figure 3.6 shows the values of $\alpha_w^i$ for each work session $h_k$, $k \in 1 \ldots 30$. We omit session $h_{13}$ (with the DIVERSITY strategy) where only 3 tasks were completed. We observe that in most work sessions, $\alpha_w^i$ oscillates around 0.5. Given the definition of $\alpha_w^i$, this value indicates that most workers do not *steadily* favor task diversity over task payment. This is particularly observable on long work sessions in Figure 3.6a, where tasks were assigned using RELEVANCE. Figure 3.7 shows the distribution of $\alpha_w^i$. It supports our observation: most workers do not make sharp choices. Most of the computed $\alpha_w^i$ values (72%) are in the interval $[0.3, 0.7]$, meaning that most workers do not favor task diversity over task payment, nor do they favor payment over diversity.

However, we do observe some sharp preferences for some workers. For instance, the


(a) RELEVANCE


(b) DIV-PAY

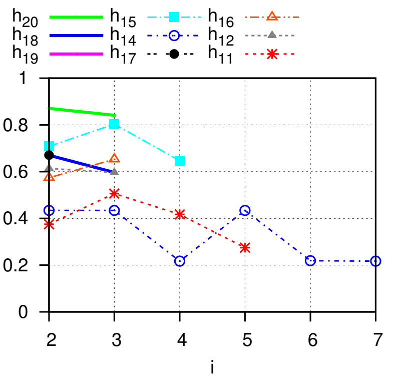

(c) DIVERSITY

Figure 3.6: Ita online experiments: evolution of $\alpha_w^i$ ($h_k$ is $k$-th work session)

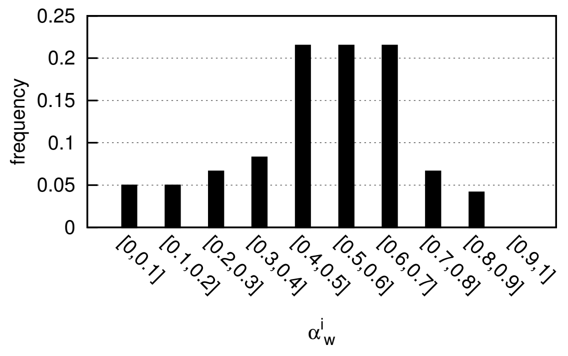


Figure 3.7: Ita online experiments: distribution of $\alpha_w^i$

worker in session $h_2$ (Figure 3.6b) clearly favored high-paying tasks. She completed 1.6 *different* tasks at each iteration (maximum possible: 5) that have an average reward of \$0.11 (maximum possible reward: \$0.12). Hence, her $\alpha_w^i$ was close to 0. Since she was assigned tasks using DIV-PAY, she received high-paying tasks. On the other hand, the worker in session $h_{25}$ (Figure 3.6a) favored task diversity (her $\alpha_w^i$ is close to 0.8). She completed 4.12 *different* tasks at each iteration, that paid \$0.05 on average. This shows that our formulation allowed to accurately capture workers' preferences between task diversity and task payment.

## 3.4 Holistic Motivation-Aware Task Assignment

In the previous section, we presented our investigations on *Individual Task Assignment* (Ita). Our experiments gave a first insight of the impact of motivation-aware task assignment on various performance dimensions. Specifically, we discovered that (i) RELEVANCE, an approach that assigns random tasks that match a worker's profile, is best for the number of completed tasks, task throughput and worker retention and (ii) that quality comes at a price: DIV-PAY offers the best crowdwork quality and pays workers the highest.

Based on these observations, we propose to explore another variant of motivation-aware task assignment. First, we aim to study how we can obtain higher workers performances if we isolate task payment and do not include it in our problem objective. We propose to keep *task diversity* and combine it with *task relevance*, since RELEVANCE showed good performance in Ita.

Second, we propose to study a different setting, where we assign tasks to *all available workers, holistically.* In Ita, we assign tasks to *one worker at a time.* However, several workers may complete tasks *simultaneously*, which is what we actually observed in our experiments in Ita. However, in Ita, workers are assigned tasks in an arbitrary order that may not enable optimal task assignment: some workers may be assigned tasks that would have best matched other workers. To optimize task assignment, we thus need to study *Holistic Task Assignment* (Hta) [102], where we assign tasks to several workers, simultaneously. Intuitively, Hta is a more complex problem than Ita since it aims at finding several sets of tasks at each iteration, one for each worker. Therefore, the performance guarantees of algorithms developed for Ita are not likely to hold on Hta. Such a difference also exists between similar pairs of problems such as between the single and multiple version of the Knapsack Problem [80] or between the variants of Maximum Dispersion [64, 65].

In this section, we first formalize the Hta problem and show that it is NP-Hard. Then we present two approximation algorithms for Hta. Finally, we conduct (i) experiments on synthetic data to verify the efficiency of our algorithms and (ii) live experiments with real workers to evaluate the performance of various assignment strategies.

### 3.4.1 Holistic Task Assignment Problem (Hta)

First, we define the expected motivation of worker $w$ on tasks $\mathcal{T}_w^i$ at iteration $i$. In Hta, we define motivation as a balance between *task diversity* and *task relevance.* We define the function $motiv_w^i$ as a linear combination of diversity and relevance of tasks in $\mathcal{T}_w^i$:

$$motiv_w^i(\mathcal{T}_w^i) = 2\alpha_w^i \times TD(\mathcal{T}_w^i) + (|\mathcal{T}_w^i| - 1) \times \beta_w^i \times TR(\mathcal{T}_w^i, w) \tag{3.10}$$

Then, we study the following problem:

**Problem 2** (Holistic Task Assignment — Hta)**.** Given a set of workers $\mathcal{W}^i \subseteq \mathcal{W}$, that are available at iteration $i$, choose $|\mathcal{W}^i|$ subsets of tasks $\mathcal{T}_w^i \subseteq \mathcal{T}^i$, one for each worker $w \in \mathcal{W}^i$, such that:

$$\begin{aligned} &\textbf{argmax} \quad \sum_{w \in \mathcal{W}^i} motiv_w^i(\mathcal{T}_w^i) \\ &\forall w \in \mathcal{W}^i,\ |\mathcal{T}_w^i| \leq X_{max} \qquad (C_1) \\ &\forall w, w' \in \mathcal{W}^i,\ \mathcal{T}_w^i \cap \mathcal{T}_{w'}^i = \emptyset \quad (C_2) \end{aligned}$$

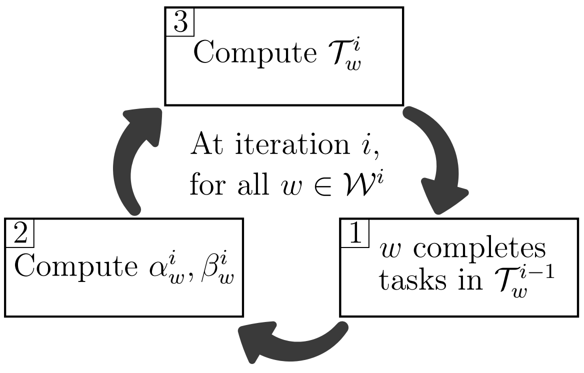

3 Compute $\mathcal{T}_w^i$
At iteration $i$, for all $w \in \mathcal{W}^i$
2 Compute $\alpha_w^i, \beta_w^i$
1 $w$ completes tasks in $\mathcal{T}_w^{i-1}$

Figure 3.8: Task assignment in Hta

Constraint $C_1$ is the same constraint as $C_2$ in Ita: it prevents task overload for workers and reflects the ability of a worker to explore only a few tasks at a time (akin to limiting Web search results). Constraint $C_2$ guarantees that each task is assigned to at most one worker. To ensure adaptive task assignment, we solve Hta per iteration. As a result, a task is assigned at most once per iteration, and tasks in $\bigcup_{w\in\mathcal{W}^i} \mathcal{T}_w^i$, once assigned, are dropped from available tasks $\mathcal{T}^{i+1}$ in the next iteration. Figure 3.8 illustrates our process.

#### 3.4.1.1 NP-Hardness

Hta is a difficult problem since it relates to several well-known NP-Hard problems. We bring here the formal proof of NP-completeness.

**Theorem 2.** *Hta is NP-complete.*

*Proof.* The decision version of Hta is as follows.

*Instance*: tasks $\mathcal{T}^i$, workers $\mathcal{W}^i$, and their $(\alpha_w^i, \beta_w^i)$, $X_{max}$ and an objective value $Z$ *Question*: are there $|\mathcal{W}^i|$ disjoint sets, $\mathcal{T}_w^i \subseteq \mathcal{T}^i$, for each worker $w$, of size $X_{max}$, such that $\sum_{w\in\mathcal{W}^i} motiv_w^i(\mathcal{T}_w^i) \geq Z$?

(1) Hta $\in NP$ : Since a nondeterministic algorithm needs only to guess $|\mathcal{W}^i|$ sets and can verify the question in polynomial time, Hta $\in$ *NP*.

(2) Hta is $NP$-hard : Let us consider the $k$-partitioning problem kPart [50]. The decision version of this problem is as follows.

*Instance*: A weighted graph $G = (V, E, \omega)$, integers $\delta$ and $k$ s.t. $\delta \geq 3$ and $|V| = k \times \delta$, an objective value $Y$; and *Question*: is there a partition of $V$ into $k$ disjoint sets $V_1, \dots, V_k$ with $|V_l| = \delta, \forall l \in [\![1, k]\!]$ s.t. $\sum_{l\in[\![1,k]\!]} \sum_{\{v_1,v_2\}\in E\cap\{V_l\times V_l\}} \omega(v_1, v_2) \geq Y$?

kPart is known to be NP-complete[3] [50] using a reduction from graph partitioning [56]. The reduction works as follows. Each vertex $v \in V$ is mapped to a task $t \in \mathcal{T}^i$. The weight of an edge $(v_1, v_2) \in E$ is mapped to skill diversity between two tasks: $\omega(v_1, v_2) =$

[3]kPart is NP-Complete even if the weights satisfy the triangle inequality [50].

$2 * d(t_{v_1}, t_{v_2})$. If there is no edge $\{v_1, v_2\} \in E$, we set $d(t_{v_1}, t_{v_2}) = 0$. We consider $|\mathcal{W}^i| = k$ highly-skilled workers who prefer skill diversity by setting for each $w \in \mathcal{W}^i$ $\alpha^i_w = 1, \beta^i_w = 0$. We set $X_{max} = \delta$ and $Z = Y$. An instance of kPart thus defines an instance of Hta where $k$ workers are assigned $\delta$ tasks and where the objective function is simplified to $2 * \sum_{w \in \mathcal{W}^i} \sum_{t_k, t_l \in \mathcal{T}^i_w\ k > l} d(t_k, t_l)$. Note that in such an instance $k \times \delta = |\mathcal{T}^i|$, therefore, all tasks are assigned. This transformation can be done in polynomial time. kPart has a solution if and only if this instance of Hta has a solution. This proves NP-hardness. □

#### 3.4.1.2 MaxSNP-Hardness

We show that in addition to being NP-complete, Hta does not admit a polynomial time approximation scheme (PTAS). Hta is related to several well-known problems that cannot be approximated in some cases. In particular, Hta is similar to the *Maximum Quadratic Assignment Problem* (MaxQap), which is known to be MaxSNP-Hard [10].

**Theorem 3.** *Hta is MaxSNP-Complete.*

*Proof.* (Sketch) To prove MaxSNP-Hardness, we use an L-reduction (linear reduction) using the maximum quadratic assignment problem (MaxQap), which is MaxSNP-Hard [10]. An L-reduction in this case is a polynomial-time mapping taking each instance $I$ of MaxQap to an instance $I'$ of Hta such that there are positive constants $a$ and $b$ that make the following statements true:

- (L1) $OPT(I') \leq a \times OPT(I)$
- (L2) For any solution of $I'$ with objective function value $OBJ'$, we can, in polynomial time, find a solution of $I$ with value $OBJ$ such that $(OPT(I) - OBJ)$ is no more than $b \times (OPT(I') - OBJ')$

We describe the maximum quadratic assignment problem (MaxQap). We adopt the formulation of Arkin et al. [10]. In MaxQap, three $M \times M$ non-negative symmetric matrices $A = (a_{k,l}), B = (b_{k,l}), C = (c_{k,l})$ are given, and the objective is to compute a permutation $\pi()$ of $\mathcal{Y} = \{1, \ldots, M\}$ that maximizes: $\sum_{k,l \in \mathcal{Y},\ k \neq l} a_{\pi(k),\pi(l)} b_{k,l} + \sum_{k \in \mathcal{Y}} c_{k,\pi(k)}$. Intuitively, when $A$ and $B$ are the adjacency matrices of two graphs, this formulation of MaxQap is equivalent to finding a subgraph in $B$ that is isomorphic to $A$ such that the total weight is maximized. We now consider an instance $I$ of MaxQap where matrices $A$, $B$ and $C$ are defined as follows. We set $A$ as the $M \times M$ adjacency matrix of $Y$ cliques of size $X$ and $M - Y * X$ isolated vertices. All edges of the same clique are labeled with the same weight. We set $B$ as the $M \times M$ adjacency matrix of a complete graph with $M$ vertices and edges labeled with a weight satisfying the triangle inequality. We set $C$ as a $M \times M$ matrix with the same values in each group of $X$ consecutive columns up to the $M - Y * X$-th column. All other values are 0. We suppose that weights $A$ and $B$ are not zero and $Z \geq 3$. Now, we transform this MaxQap instance $I$ to an instance $I'$ of Hta using the following steps: (1)

We set $X_{max} = X$; (2) We set $|\mathcal{T}^i| = M$ tasks that satisfy $\forall k, l \in 1, \ldots, |\mathcal{T}^i|\ d(t_k, t_l) = b_{k,l}$; (3) We set $|\mathcal{W}^i| = Y$ workers and $\forall q \in 1, \ldots, |\mathcal{W}^i|\ \alpha^i_{w_q} = a_{X_{max}(q-1)+1, X_{max}(q-1)+1}, \beta^i_{w_q} = c_{1, X_{max}*(q-1)+1}/(X_{max} - 1)$ and, $\forall k \in 1, \ldots, |\mathcal{T}^i|\ rel(w_q, t_k) = 1$; (4) We set $\mathcal{T}^i_{w_q} = \{t_k \mid \lceil \pi(k)/X_{max} \rceil = q\}$. Clearly, such a transformation is done in polynomial time. Now, we present two lemmas that are crucial for the proof.

**Lemma 1.** *$OPT(I') = a \times OPT(I)$, where $a = 1$.*

**Lemma 2.** *$(OPT(I) - OBJ) = b \times (OPT(I') - OBJ')$, where $b = 1$.*

We omit the proofs of both lemmas for brevity and refer to Section 3.4.2.2 where we show that the values of the objective functions of these two seemingly different problems are actually identical (Equation 3.17). Therefore, given an instance of $I$ of Hta, its optimum objective function value is the same as that of $I'$ of MaxQap. In addition, the gap in the objective function value between the optimum and any other solution is the same for both problems. From these two aforementioned lemmas, we verify the conditions specified in (L1) and (L2) above that are necessary for the proof. $\square$

### 3.4.2 Our Approach for Hta

In this section, we present our two main components for motivation-aware task assignment. First, we present how we capture the expected motivation of a worker. We show how we define $\alpha^i_w$ and $\beta^i_w$ at each iteration.

Then, we investigate task assignment. Since Hta is NP-hard and even hard to approximate, it is prohibitively expensive to solve on large instances. In our scenario, response time is important since task assignment has to be solved *online*, at each iteration $i$. The challenge is to design efficient algorithms that also have provable factors. We propose two algorithms. They both rely on the assumption that the distance function used to model diversity is a metric. That is not an overstretch as Jaccard is indeed a metric [16]. When that property is not assumed, Hta remains largely inapproximable [10, 19, 93]. Our algorithms have provable approximation factors - the first one has a better approximation factor but incurs a higher running time. The second compromises slightly on the approximation factor to ensure a faster running time.

#### 3.4.2.1 Computing $\alpha^i_w$ and $\beta^i_w$

In Section 3.2, we defined how to capture the importance of each factor (Equations 3.4 and 3.6) when a worker chooses a task in the set of available tasks. We now need to define the two parameters $\alpha^i_w$ and $\beta^i_w$. Suppose that during iteration $i-1$ worker $w$ chose $J$ tasks where $J \leq |\mathcal{T}^{i-1}_w|$. We define $\alpha^i_w$ and $\beta^i_w$ as follows:

$$\alpha^i_w = \underset{j \in [\![2, J]\!]}{\text{avg}}\ \Delta TD(t_j) \tag{3.11}$$

$$\beta^i_w = \underset{j\in[\![2,J]\!]}{\text{avg}} \Delta TR(t_j, w) \tag{3.12}$$

Equations 3.11-3.12 are just the average of each observation made when $w$ was completing tasks in $\mathcal{T}^{i-1}_w$. We skip the case where $j = 1$ that corresponds to task completed first, since the marginal gain in task diversity would return 0 (Equation 3.4).

#### 3.4.2.2 MaxQap and Hta

Before presenting the actual algorithms, we show that the objective functions of MaxQap and Hta are actually identical. We first describe the maximum quadratic assignment problem (MaxQap) using the formulation of Arkin et. al[10]. In MaxQap, three $M \times M$ non-negative symmetric matrices $A = (a_{k,l}), B = (b_{k,l}), C = (c_{k,l})$ are given, and the objective is to compute a permutation $\pi()$ of $\mathcal{Y} = \{1, \ldots, M\}$ that maximizes:

$$\sum_{k,l\in\mathcal{Y},\ k\neq l} a_{\pi(k),\pi(l)} b_{k,l} + \sum_{k\in\mathcal{Y}} c_{k,\pi(k)}$$

MaxQap has several special cases that are popular problems [10, 19, 93], especially when $A$ and $B$ are adjacency matrices of graphs.

Recall that Hta admits tasks $\mathcal{T}^i = \{t_1, \ldots, t_{|\mathcal{T}^i|}\}$ and workers $\mathcal{W}^i = \{w_1, \ldots, w_{|\mathcal{W}^i|}\}$. First, we define $A$ to be the $|\mathcal{T}^i| \times |\mathcal{T}^i|$ adjacency matrix of $|\mathcal{W}^i|$ disjoint cliques of $X_{max}$ vertices and $|\mathcal{T}^i| - |\mathcal{W}^i| * X_{max}$ isolated vertices. Each clique corresponds to a worker. In each clique, edges are labeled with $\alpha^i_w$. Thus, in the graph whose adjacency matrix is $A$, each worker is mapped to a set of vertices. We set $B$ to be the adjacency matrix of the complete graph where vertices are mapped to tasks in $\mathcal{T}^i$ and edges are labeled with pairwise task diversities. We define the $|\mathcal{T}^i| \times |\mathcal{T}^i|$ matrix $C$ to represent the linear part of our objective function, i.e., the relevance part. If $l$ is a column that corresponds to a worker $w$ in matrix $A$, then $c_{k,l}$ equals $rel(w, t_k)$ multiplied by $\beta^i_w * (X_{max} - 1)$. Formally:

$$\forall k, l \in 1, \ldots, |\mathcal{T}^i|,\ a_{k,l} = \begin{cases} \alpha^i_{w_{\lceil l/X_{max}\rceil}} & \text{if } \lceil l/X_{max}\rceil \leq |\mathcal{W}^i| \wedge \lceil k/X_{max}\rceil = \lceil l/X_{max}\rceil \\ 0 & \text{else} \end{cases} \tag{3.13}$$

$$\forall k, l \in 1, \ldots, |\mathcal{T}^i|,\ b_{k,l} = d(t_k, t_l) \tag{3.14}$$

$$\forall k, l \in 1, \ldots, |\mathcal{T}^i|,\ c_{k,l} = \begin{cases} \beta^i_{w_{\lceil l/X_{max}\rceil}} rel(w_{\lceil l/X_{max}\rceil}, t_k)(X_{max} - 1) & \text{if } l \leq |\mathcal{T}^i| - |\mathcal{W}^i| * X_{max} \\ 0 & \text{else} \end{cases} \tag{3.15}$$

| $rel(t,w)$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ |
|---|---|---|---|---|---|---|---|---|
| $w_1$ | 0.28 | 0.25 | 0.2 | 0.43 | 0.67 | 0.4 | 0 | 0.4 |
| $w_2$ | 0.3 | 0 | 0.2 | 0.25 | 0.25 | 0 | 0 | 0.4 |

Table 3.3: Hta: example of tasks and workers

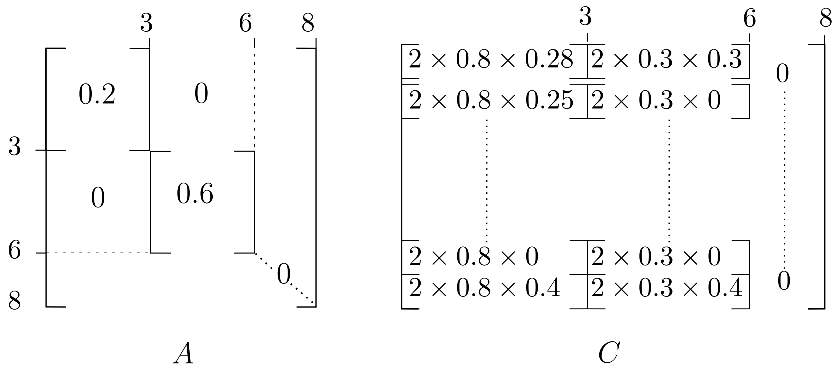


Figure 3.9: Example of mapping to MaxQap: matrices $A$ and $C$

*Example* 3. Suppose that we have 2 workers and 8 tasks as exposed in Table 3.3. Let $X_{max} = 3$, $\alpha^i_{w_1} = 0.2$, $\beta^i_{w_1} = 0.8$, $\alpha^i_{w_2} = 0.6$ and $\beta^i_{w_1} = 0.3$. Figure 3.9 shows matrices $A$ and $C$, as defined by Equations 3.13 and 3.15.

Given that $A$ and $B$ are adjacency matrices, solving our MaxQap instance is equivalent to finding a subgraph in $B$ that is isomorphic to $A$ such that the objective function is maximized [10]. Indeed, a permutation $\pi$ of $\{1,\dots,|\mathcal{T}^i|\}$ maps each vertex of $B$ to a vertex of $A$. In our settings, this maps a task to a vertex in $A$, which may be associated to a worker. Then, let us consider a pair of tasks $(t_k, t_l)$ that is assigned to the pair of vertices $(\pi(k), \pi(l))$. The profit induced by this pair is $a_{\pi(k),\pi(l)} * b_{k,l} + c_{k,\pi(k)} + c_{l,\pi(l)}$. Thus, if a worker $w_q$ is associated to both vertices $\pi(k)$ and $\pi(l)$ in $A$, the profit induced by this pair is:

$$\alpha^i_{w_q} * d(t_k, t_l) + \beta^i_{w_q}(X_{max} - 1)(rel(w_q, t_k) + rel(w_q, t_l))$$

The left part $\alpha^i_{w_q} * d(t_k, t_l)$ corresponds to task diversity. Since $A$ is composed of $|\mathcal{W}^i|$ cliques and given the objective function, $\alpha^i_{w_q} * d(t_k, t_l)$ is counted for each pair of tasks that is associated to the same worker (*if* clause in Equation 3.13). The right part $\beta^i_{w_q}(X_{max} - 1)(rel(w_q, t_k) + rel(w_q, t_l))$ corresponds to task relevance. It is counted for each task that is associated to a worker (*if* clause in Equation 3.15).

Now, we can map the output of the MaxQap instance to the output of Hta. For all workers $w_q$ in $\mathcal{W}^i$, we set:

$$\mathcal{T}^i_{w_q} = \{t_k \mid \lceil \pi(k)/X_{max} \rceil = q\} \tag{3.16}$$

**Algorithm 9** Hta-App

```
Input: $\mathcal{W}^i = \{w_1, \ldots, w_{|\mathcal{W}^i|}\}$, $\mathcal{T}^i = \{t_1, \ldots, t_{|\mathcal{T}^i|}\}$
Output: $|\mathcal{W}^i|$ sets of tasks $\{\mathcal{T}^i_{w_1}, \ldots, \mathcal{T}^i_{w_{|\mathcal{W}^i|}}\}$
 1: Build matrices $A, B, C$ (Equations 3.13, 3.14 and 3.15)
 2: $M_B \leftarrow$ maximum weight matching in $B$
 3: for $k \in 1, \ldots, |\mathcal{T}^i|$
 4:     $deg^A_k = \sum_{l \in 1,\ldots,|\mathcal{T}^i| \setminus \{k\}} a_{k,l}$
 5:     if $t_k$ is covered by $M_B$
 6:         $b_M(t_k) =$ weight of the edge in $M_B$ incident to $t_k$
 7:     else
 8:         $b_M(t_k) = 0$
 9: for $k, l \in 1, \ldots, |\mathcal{T}^i|$
10:     $f_{k,l} = b_M(t_k) deg^A_l + c_{k,l}$
11: $\pi' \leftarrow$ an optimal solution to the linear assignment problem $\max_\sigma \sum_k f_{k,\sigma_k}$
12: for $(t_k, t_l) \in M_B$
13:     $\pi(k) = \pi'(k)$ and $\pi(l) = \pi'(l)$ with probability $1/2$
14:     $\pi(k) = \pi'(l)$ and $\pi(l) = \pi'(k)$ otherwise
15: if $k \notin M_B$
16:     $\pi(k) = \pi'(k)$
17: for $w_q \in \mathcal{W}^i$
18:     $\mathcal{T}^i_{w_q} \leftarrow \{t_k \mid \lceil \pi(k)/X_{max} \rceil = q\}$
19: return $\{\mathcal{T}^i_{w_1}, \ldots, \mathcal{T}^i_{w_{|\mathcal{W}^i|}}\}$
```

Equation 3.16 partitions $\mathcal{T}^i$ into $|\mathcal{W}^i|$ sets using the solution $\pi$ of the MaxQap instance. We obtain a solution for Hta by solving its corresponding MaxQap instance.

*Example* 4. We continue with Example 3. Suppose that the solution of the MaxQap instance returns $\pi(1) = 4, \pi(4) = 1$ and $\forall k \in 1, \ldots, |\mathcal{T}^i|, k \notin \{1, 4\}$ $\pi(k) = k$. Given Equation 3.16, we have $\mathcal{T}^i_{w_1} = \{t_4, t_2, t_3\}$ and $\mathcal{T}^i_{w_2} = \{t_1, t_5, t_6\}$. Tasks $t_7$ and $t_8$ are left unassigned.

Following the previous observations, we can show that:

$$\sum_{w \in \mathcal{W}^i} motiv^i_w(\mathcal{T}^i_w) = \sum_{\substack{k,l \in 1,\ldots,|\mathcal{T}^i| \\ k \neq l}} a_{\pi(k),\pi(l)} b_{k,l} + \sum_{k \in 1,\ldots,|\mathcal{T}^i|} c_{k,\pi(k)} \tag{3.17}$$

This shows that the value of the objective function of our Hta instance is same as the MaxQap instance that we built. We prove Equation 3.17 in Appendix B.1.

#### 3.4.2.3 Approximation Algorithm Hta-App

In Section 3.4.2.2, we showed how we can map an instance of Hta to an instance of MaxQap. Here, we adapt to our settings the algorithm of Arkin et al. [10] that was designed for MaxQap. We present Hta-App (Algorithm 9). First Hta-App maps tasks and workers to matrices (Line 1 in Algorithm 9). Matrix $A$ is the weighted adjacency matrix of $|\mathcal{W}^i|$ cliques (one per worker). Matrix $B$ is the weighted adjacency matrix of the graph where vertices are mapped to tasks and edges are labeled with their pairwise task diversity. Matrix $C$ represents the task relevance part, where each row represents a task and each column a possible assignment for this task. Then, from Lines 2 to 16, Hta-App uses only these matrices. First, it finds a maximum weight matching $M_B$ with respect to task diversity using matrix $B$. This step identifies a set of task pairs whose aggregated diversity value is maximized. Then, Hta-App builds an auxiliary problem using $M_B$ and matrices $A$ and $C$. This problem is an instance of the *Linear Sum Assignment Problem*(Lsap) [19]. On Line 10, Hta-App combines the profit associated to task diversity and task relevance. Observe that if a vertex $k$ is not associated to a worker in matrix $A$, then $\forall k \in 1, \ldots, |\mathcal{T}^i|$ $f_{k,l} = 0$. If $k$ is associated to worker $w_q$ and if task $t_k$ is incident to the edge $(t_k, t_l) \in M_B$, we have $f_{k,l} = d(t_k, t_l) * (X_{max} - 1) * \alpha^i_{w_q} + \beta^i_{w_q} * (X_{max} - 1) * rel(w_q, t_k)$. The rationale behind using this auxiliary problem with those weights is to build a *linear* problem that is easier to solve since our original problem is inherently *quadratic*. Intuitively, $f_{k,l}$ is an approximated profit obtained when assigning task $t_k$ to vertex $l$. On Line 11, Hta-App solves the Lsap instance. Then, it uses a slightly modified version of the solution $\pi'$ of the Lsap problem. For each edge $(t_k, t_l)$ in the maximum matching $M_B$, it randomly permutes the assignation of $t_k$ and $t_l$ (Lines 12-14). If a task is not in the matching, it is not permuted (Lines 15-16) further. Finally, Hta-App builds a solution for Hta using Equation 3.16 (Lines 17-18).

*Example* 5. We run Hta-App using workers and tasks from Example 3. Suppose that $M_B = \{(t_4, t_8), (t_1, t_6), (t_3, t_2), (t_7, t_5)\}$ and $d(t_4, t_8) = 1, d(t_1, t_6) = 1, d(t_3, t_2) = 0.86, d(t_7, t_5) = 0.8$ (Algorithm 9 - Line 2). Then, Hta-App will set $f_{1,1} = 1*0.4+0.448 = 0.848$. This profit is used in the Lsap instance on Line 11. It is an approximation of the profit obtained when assigning task $t_1$ to vertex 1, which is associated to worker $w$. Then, Hta-App solves Lsap and we obtain $\pi = (4, 7, 1, 6, 3, 8, 2, 5)$ (Line 16). Therefore, worker $w_1$ is assigned tasks $t_3, t_5, t_7$ and worker $w_2$ tasks $t_1, t_4, t_8$.

**Theorem 4.** Hta-App *is a $\frac{1}{4}$-approximation algorithm for the Hta problem.*

*Proof.* We use an approximation-preserving reduction to prove the approximation factor [11]. In Section 3.4.2.2, we exposed how to map any instance of Hta to an instance of MaxQap. We also showed that an optimal solution for the obtained instance of MaxQap is an optimal solution for Hta. With this one to one mapping between Hta and MaxQap, an approximation algorithm for MaxQap will also solve Hta with the same approximation guarantee. Hta-App is an adaptation of the approximation algorithm proposed by Arkin

**Algorithm 10** HTA-GRE

```
Input: $\mathcal{W}^i = \{w_1, \ldots, w_{|\mathcal{W}^i|}\}$, $\mathcal{T}^i = \{t_1, \ldots, t_{|\mathcal{T}^i|}\}$
Output: $|\mathcal{W}^i|$ sets of tasks $\{\mathcal{T}^i_{w_1}, \ldots, \mathcal{T}^i_{w_{|\mathcal{W}^i|}}\}$
 1: Build matrices $A, B, C$ (Equations 3.13, 3.14 and 3.15)
 2: $M_B \leftarrow$ maximum weight matching in $B$
 3: ... (Lines 3-10 of Algorithm 9)
11: $\pi' \leftarrow$ greedy matching in the completed bipartite graph associated to the Lsap problem
    $\max_\sigma \sum_k f_{k,\sigma_k}$
12: ... (Lines 12-18 of Algorithm 9)
19: return $\{\mathcal{T}^i_{w_1}, \ldots, \mathcal{T}^i_{w_{|\mathcal{W}^i|}}\}$
```

et al [10] (Algorithm 9 - Lines 2-16), which is proved to have an $\frac{1}{4}$-approximation factor for the MaxQap problem. Therefore, HTA-APP is also a $\frac{1}{4}$-approximation for Hta[4]. $\square$

**Lemma 3.** HTA-APP *runs in* $\mathcal{O}(|\mathcal{T}^i|^3)$ *time*

*Proof.* The cubic time complexity comes from the linear assignment problem that the algorithm needs to solve (Algorithm 9 - Line 11). This kind of problem is typically solved using the Hungarian algorithm that runs in $\mathcal{O}(|\mathcal{T}^i|^3)$ time [19]. The matching step (Line 2) can be done in $\mathcal{O}(|\mathcal{T}^i|^2 \log |\mathcal{T}^i|)$ time using a greedy matching [10]. $\square$

#### 3.4.2.4 Approximation Algorithm HTA-GRE

Although HTA-APP tackles the complexity of our problem by returning, in polynomial time, a solution that verifies a performance guarantee, its running time may not be satisfactory. We now propose an alternative algorithm HTA-GRE that runs in $\mathcal{O}(|\mathcal{T}^i|^2 \log |\mathcal{T}^i|)$ time, where the approximation factor is slightly compromised.

In Lemma 3, we observed that the time complexity of HTA-APP is dominated by the time complexity of the algorithm that solves the Lsap problem (Algorithm 9 - Line 11). This step can be done in $\mathcal{O}(|\mathcal{T}^i|^3)$ time, using improved versions of the well-known Hungarian algorithm [19]. To the best of our knowledge $\mathcal{O}(|\mathcal{T}^i|^3)$ is the best *polynomial* time complexity we can obtain to solve this assignment step. There also exists a number of algorithms that solve Lsap in *pseudo-polynomial* time [19] (e.g. *cost-scaling* algorithms that run in $\mathcal{O}(|\mathcal{T}^i|^{\frac{5}{2}} \log(|\mathcal{T}^i|\mathcal{C}))$ time where $\mathcal{C} = \max_{k,l \in 1,\ldots,|\mathcal{T}^i|} f_{k,l}$). Since our goal is to reach a *polynomial* time approximation, those algorithms are not applicable. Our proposed HTA-GRE algorithm (Algorithm 10) improves the cubic running time complexity. Our rationale is the following: rather than solving *optimally* the Lsap problem, we aim to find an *approximate* solution in polynomial time (Algorithm 10 - Line 11). Specifically, we find

[4]The performance guarantee holds even if the matrix $C$ is not symmetric: the proof of Arkin et al. [10] only relies on the symmetry of $A$ and $B$.

**Algorithm 11** GreedyMatching

```
Input: $\mathcal{G} = (V, E), w : E \to \mathbb{R}^+$
Output: $\mathcal{M}$ greedy matching on $\mathcal{G}$
1: $\mathcal{M} \leftarrow \emptyset$
2: while $E \neq \emptyset$
3:     $e \leftarrow$ heaviest edge in $E$
4:     $\mathcal{M} \leftarrow \mathcal{M} \cup e$
5:     remove $e$ and all edges incident to $e$ from $E$
6: return $\mathcal{M}$
```

a greedy matching in the complete weighted bipartite graph associated to the Lsap instance. The rest of the algorithm remains unchanged. In the next paragraph, we describe how we find this approximation. Then, we show that Hta-Gre is a $\frac{1}{8}$-approximation algorithm that runs in $\mathcal{O}(|\mathcal{T}^i|^2 \log |\mathcal{T}^i|)$ time.

**Approximation for Lsap.** We present an approximation algorithm to solve Lsap. For that, we first redefine the Lsap problem using graph theory [19]. Lsap can be modeled using a weighted complete bipartite graph $\mathcal{G}^{\mathsf{Lsap}} = (U, V, E, f)$. We set $U = V = \{1, \ldots, |\mathcal{T}^i|\}$. Each edge $e = (k, l),\ k \in U, l \in V$ has the weight $f_{k,l}$. Solving Lsap becomes equivalent to solving a *Maximum Weight Perfect Matching Problem* (Mwpmp) [19, 40]. In graph theory, the problem of finding a matching, i.e., a set of vertex-disjoint edges, with the maximal weight is called the *maximum weight matching problem* (Mwmp) [40]. A *maximum weight perfect matching* is a maximum weight matching that covers all vertices. To solve Mwpmp, we employ the well-known GreedyMatching algorithm [40] (Algorithm 11), that selects the heaviest edge $e \in E$, removes $e$ and edges incident to $e$, and reiterates until no edges are left. It is well-known that GreedyMatching is a $\frac{1}{2}$-approximation for Mwmp [38, 40]. We just need to show that this performance guarantee holds for the Mwmp on $\mathcal{G}^{\mathsf{Lsap}}$.

**Lemma 4.** GreedyMatching *is a $\frac{1}{2}$-approximation for the Mwmp on $\mathcal{G}^{\mathsf{Lsap}}$.*

*Proof.* (sketch) Because $\mathcal{G}^{\mathsf{Lsap}}$ is complete and counts an even number of vertices ($2|\mathcal{T}^i|$), it is easy to see that GreedyMatching returns a set of edges that cover all vertices. Thus, GreedyMatching returns a perfect matching. This completes the proof. □

**Performance guarantee of Hta-Gre.** We show that Hta-Gre is a $\frac{1}{8}$-approximation for the Hta problem.

**Theorem 5.** Hta-Gre *algorithm is a $\frac{1}{8}$-approximation for the Hta problem.*

*Proof.* (sketch): The proof structure is analogous to that of the approximation preserving reduction presented to prove Theorem 4. Once an instance of Hta is reduced to an instance of MaxQap, the proof follows two parts. First, we show that the value of the optimal solution for Hta is less than 4 times the solution value for the auxiliary problem Lsap. Then, we show that the value of the solution returned by Hta-Gre is greater than $\frac{1}{2}$ times the value of the optimal solution for Lsap. Appendix B.2 contains further details. Combining both propositions completes the proof. □

**Lemma 5.** *The running time complexity of* Hta-Gre *is* $\mathcal{O}(|\mathcal{T}^i|^2 \log |\mathcal{T}^i|)$.

*Proof.* The time complexity of Hta-Gre is dominated by its two matching steps (Algorithm 10 - Lines 2 and 11). As observed by Arkin et al., the first matching step (Line 2) can be completed using a *greedy* matching. It is well-known that a greedy matching can be computed in $\mathcal{O}(|E| \log |V|)$ time on a graph $G = (V, E)$. The first matching runs on $|\mathcal{T}^i|$ vertices and $\frac{|\mathcal{T}^i|(|\mathcal{T}^i|-1)}{2}$ edges. The second matching (Line 11) runs on $2|\mathcal{T}^i|$ vertices and on $|\mathcal{T}^i|^2$ edges. The overall running time complexity is thus $\mathcal{O}(|\mathcal{T}^i|^2 \log |\mathcal{T}^i|)$. □

## 3.4.3 Experiments

We run two kinds of experiments: (1) an offline simulation to stress the performance of our algorithms, and (2) an online deployment where we capture workers' motivation, perform task assignment accordingly, and measure end-to-end performance.

### 3.4.3.1 Summary of Results

Our first set of experiments is based on simulated workers and evaluates scalability and expected motivation, i.e., value of objective function. We find that Hta-Gre performs better than Hta-App thanks to its greedy strategy that depends mainly on the number of tasks, *not* on the number of workers. The Hungarian algorithm on which Hta-App relies is, on the other hand, more expensive. We find that Hta-Gre has an acceptable response time and could hence be executed in the background while workers complete tasks, to prepare the next round of assignments. We also find that the greedy strategy of Hta-Gre does not negatively affect the value for the objective function when compared to Hta-App. Therefore, Hta-Gre should be the algorithm of choice for the holistic task assignment problem as it runs faster while producing an assignment comparable to Hta-App.

Our second experiments rely on effectively deploying tasks with real workers. We compare Hta-Gre with three other non-adaptive strategies: a random assignment Random, one based on diversity only Hta-Gre-Div and another on relevance only Hta-Gre-Rel. We first show that optimizing diversity only (Hta-Gre-Div) results in the highest crowdwork quality and relevance only (Hta-Gre-Rel) is worst. We can therefore conjecture that diversity addresses boredom. In fact, relevance only is consistently outperformed along quality, number of completed tasks and retention. We also find that Hta-Gre offers the best compromise between those dimensions thereby assessing the need for adaptability.

#### 3.4.3.2 Offline Simulation

We focus on one iteration at a time and measure scalability in terms of how fast tasks are assigned, and motivation, in terms of the objective function. Ideally, a task assignment algorithm should be quick and should maximize workers' motivation. We compare the performance of Hta-App and Hta-Gre.

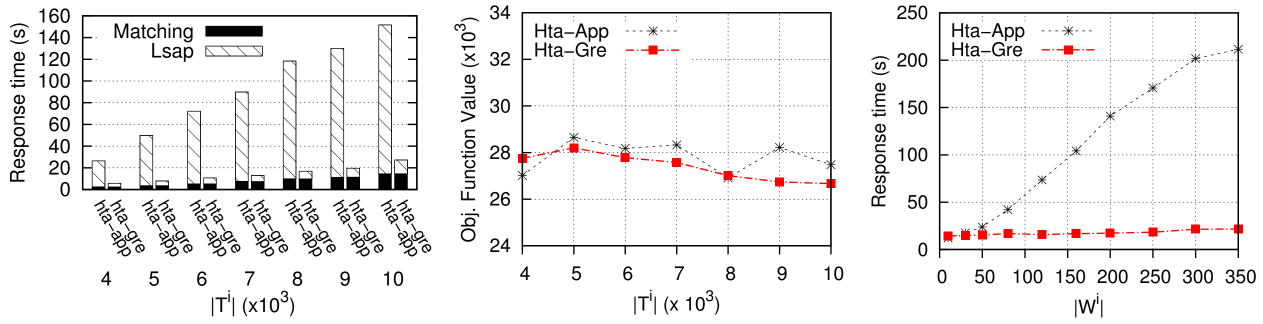
**Implementation.** We implement Hta-App and Hta-Gre in Java and run them on Oracle's 1.8.0_91 JVM on a Debian 8.7 server with 2 Intel Xeon E5-2650@2.60 GHz and 128GB of RAM. For all sorting sub-routines, specifically when a *greedy* matching has to be computed, we use `Arrays.sort`, an improved implementation of the Merge Sort algorithm (know as "TimSort", runs in $\mathcal{O}(n \log n)$ time). To solve `Lsap` in Hta-App, we adapt the C implementation of the Hungarian Algorithm of Carpaneto et al. [24] which runs in $\mathcal{O}(n^3)$ time (available online [19]). We report the average of 10 runs each time.

**Data Sets.** We use real tasks and synthetic workers. We crawled $152,221$ *task groups* from Amazon Mechanical Turk (AMT). Each task group contains metadata about tasks in that group. We vary the size and the features of groups and tasks, as exposed in Appendix A.1. Our workers are synthetically generated. For each worker $w$, we use a pseudo-random uniform generator to choose 5 keywords and we pick a random $\alpha_w^i$ and $\beta_w^i$ in the range of $[0, 1]$.

**Results.** We conduct experiments where we vary three dimensions: (i) the number of tasks $|\mathcal{T}^i|$, (ii) the number of workers $|\mathcal{W}^i|$ and (iii) the diversity of tasks in $\mathcal{T}^i$.

**Number of Tasks.** We vary the number of tasks $|\mathcal{T}^i|$ from $4,000$ to $10,000$, with 200 tasks per task group. Figure 3.10a shows the detailed response times of Hta-App and Hta-Gre. As expected, Hta-App is outperformed by Hta-Gre as its response time grows faster with the number of tasks. We can observe that the difference comes from the second phase of the algorithm that solves the auxiliary problem `Lsap`. For this phase, the response time of Hta-App is $\mathcal{O}(|\mathcal{T}^i|^3)$ while Hta-Gre is $\mathcal{O}(|\mathcal{T}^i|^2 \log |\mathcal{T}^i|)$. The results in Figure 3.10b show that both Hta-App and Hta-Gre report very similar values for the objective function, confirming the benefit of Hta-Gre.

**Number of Workers.** Figure 3.10c shows the results response time as a function of the number of workers. Here again, Hta-Gre outperforms Hta-App. The difference comes from the Hungarian Algorithm in Hta-App whose response time increases with the number of workers. That is due to the number of *0-weight* edges used in its initialization phase. The more 0-weight edges, the higher the chance to find an initial solution that covers all vertices. This allows early termination and avoids running the expensive sub-routine `augment` to find an augmenting path [19] (which runs in $\mathcal{O}(|\mathcal{T}^i|^2)$ time). Even if the



(a) X-axis: Number of tasks $|\mathcal{W}^i| = 200$ (b) X-axis: Number of tasks $|\mathcal{W}^i| = 200$ (c) X-axis: Number of workers $|\mathcal{T}^i| = 8000$

200 task groups, $X_{max} = 20$

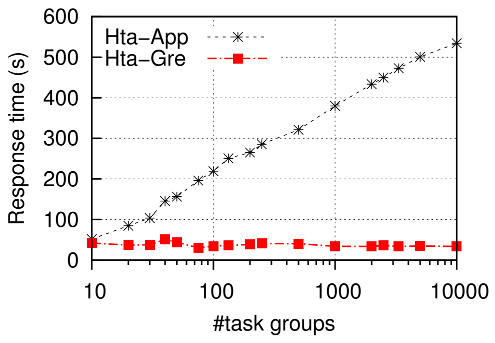
Figure 3.10: Hta synthetic experiments: scalability w.r.t. the number of tasks and workers

initial solution does not cover all vertices, more 0-weight edges also increase the chances of finding a long alternating path, favoring early termination. For instance, the number of calls to the `augment` sub-routine was 66 times higher when $|\mathcal{W}^i| = 350$ than when $|\mathcal{W}^i| = 30$. This is because many edges have the same weight when there are fewer workers (i.e. there are many $k, l \in 1, \ldots, |\mathcal{T}^i|$ where $f_{k,l} = 0$), which in turn incurs 0-values in the dual problem. The sub-routine GreedyMatching in Hta-Gre was less affected by the number of workers since the response time of the sorting phase in GreedyMatching increases slowly with the number of workers. We also observed similar results for the value of the objective function.

**Diversity of Tasks.** We vary the number of task groups from 10 to 10,000 with a fixed number of tasks $|\mathcal{T}^i| = 10,000$ (e.g. when there are 10,000 task groups there is one task per task group). Similarly to earlier, Hta-Gre outperforms Hta-App mainly because of the Hungarian Algorithm in Hta-App (Figure 3.11). The more diverse the tasks, the more diverse the $f_{k,l}$ values in the Lsap problem, and the smaller the chance to build 0-weight edges in the dual problem. Therefore, the higher the number of diverse tasks, the less likely the Hungarian Algorithm will terminate early. Our algorithm Hta-Gre is oblivious to the diversity of tasks. Indeed, the running time of the sort phase in the GreedyMatching sub-routine does not depend on the diversity of $f_{k,l}$ values.

### 3.4.3.3 Online deployment

The purpose of online deployment is to verify the behavior of Hta-Gre in practice and assess the utility of diversity to model motivation, and the need for adaptability with real workers. Each worker enters a work session during which several iterations occur. Similarlty to Ita, we measure three main performance indicators: *outcome quality* that



$|\mathcal{T}^i| = 10^3$, $|\mathcal{W}^i| = 300$, $X_{max} = 20$

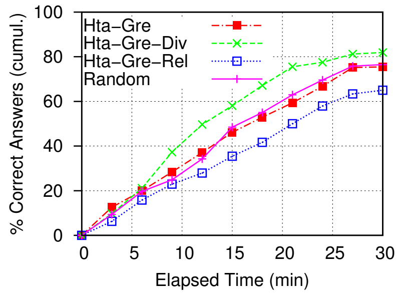
Figure 3.11: Hta synthetic experiments: effect of task diversity

reflects the quality of a worker's contribution to a task with regard to a ground truth, *number of completed tasks* that measures the number of completed tasks per session and per unit of time, and *worker retention* that characterizes how long workers stay motivated during a work session.

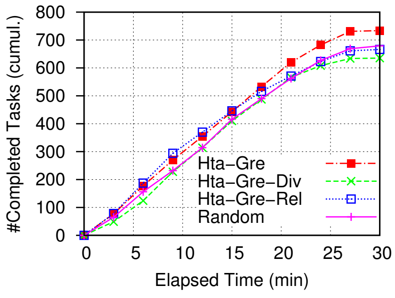
**Set-up.** We compare Hta-Gre with 3 other assignment strategies: (i) Random, that assigns a set of $X_{max}$ random tasks, (ii) Hta-Gre-Rel, that uses Hta-Gre and sets $\alpha^i_w = 0, \beta^i_w = 1 \, \forall w \in \mathcal{W}^i$ hence optimizing relevance only, and (iii) Hta-Gre-Div, that uses Hta-Gre with $\alpha^i_w = 1, \beta^i_w = 0 \, \forall w \in \mathcal{W}^i$ hence optimizing diversity only. Hta-Gre is the only algorithm that is adaptive: it needs to compute $\alpha^i_w$ and $\beta^i_w$ for each worker in $\mathcal{W}^i$. For the first iteration of Hta-Gre, we assign random tasks using Random to address the cold start.

**Workflow.** We employ our platform GACS, where workers hired from AMT can complete tasks (see the workflow in GACS in Appendix A.2). One notable difference with Ita is that Hta require to monitor *all workers at once*. A new assignment iteration occurs when at least 5 tasks are completed *overall* and when at least one worker completes a minimum of 8 tasks. We set $X_{max} = 15$ and display a total of 20 tasks per worker: we add 5 random tasks to avoid falling into a silo by assigning only highly relevant tasks to each worker.

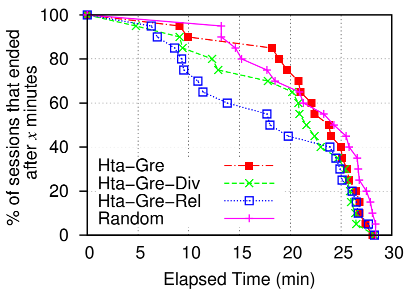
**Tasks and Workers.** We used a set of 158, 018 micro-tasks released by Crowdflower. It includes 22 different kinds of tasks. Appendix A.1 presents this dataset in details. We ran our experiments using a sample of 5, 000 tasks where each kind of task is equally represented. We published 160 HITs on AMT to recruit workers. The HIT reward is set to \$0.1 and workers receive a bonus equivalent to the total reward of tasks that they completed. To qualify for our experiment, we required workers to have previously completed at least 100 HITs that were approved, and to have an approval rate above 80%. We also required HITs to be completed within 30 minutes. Then, we filtered results. We omit work sessions



(a) Crowdwork quality



(b) Number of completed tasks



(c) Worker retention

| Strategy | % Correct answers | Avg. # completed tasks/session | Avg. session duration (min.) | # Completed tasks/min | Total task payment ($) | Avg. task payment ($ cents) |
|---|---|---|---|---|---|---|
| Hta-Gre | 75.5% | 36.7 | 22.3 | 1.6 | 46.8 | 6.4 |
| Hta-Gre-Div | 81.9% | 31.8 | 20.2 | 1.6 | 31.1 | 4.9 |
| Hta-Gre-Rel | 65.0% | 33.3 | 18.3 | 1.8 | 37.3 | 5.6 |
| Random | 76.5% | 34.0 | 22.7 | 1.5 | 46.0 | 6.8 |

(d) Aggregated measures

Figure 3.12: Hta online experiments: results

where workers did not follow instructions or complete enough tasks to be re-assigned new tasks[5]. In order to make our strategies comparable, we selected, for each strategy, the 20 work sessions with the highest number of completed tasks. Overall, 58 different workers completed 2,715 tasks in 80 work sessions (4 strategies).

**Results.** We collected three main performance measures: (i) crowdwork quality, that measures the quality of workers' contributions, (ii) number of completed tasks and (iii) worker retention, that reflects how workers are prone to stay longer on our platform to complete more tasks.

**Quality.** We measure the quality of individual contributions from the crowd. 4,473 questions were asked for the 2,715 tasks that were completed (a task may have several questions). For a sample of 1,137 questions, we used the ground truth provided by Crowdflower. Figure 3.12a presents the cumulative percentage of questions that were correctly completed for each strategy (see Table 3.12d for the aggregated measures). We observe that workers performed better with Hta-Gre-Div (81.9% of correct answers) than with other strategies. This is confirmed at the significance level 0.06 using two-proportions Z-test on the proportions of correct answers in each strategy. Hta-Gre and Random were slightly below (75.5% and 76.5%) and outperform Hta-Gre-Rel (significance level: 0.01) that has

[5]Some workers left after one task. Others stayed for over 3 hours!

the worst ratio (65%). In fact, with Hta-Gre, the rate at which correct answers are provided is constant, while it starts to drop for Hta-Gre-Rel after 21 minutes. This result validates our choice of using diversity in the expression of workers' motivation. It raises however the utility of adaptability. *If workers perform best along the quality dimension with* Hta-Gre-Div*, what do we need* Hta-Gre *for?*

**Number of completed tasks.** Figure 3.12b presents the cumulative number of completed tasks for each strategy (see Table 3.12d for the aggregated measures). Overall, Hta-Gre outperforms all others: 734 tasks were completed with Hta-Gre, 679 with Random, 666 with Hta-Gre-Rel and 636 with Hta-Gre-Div. The fact that Hta-Gre outperformed Hta-Gre-Div is confirmed at the significance level 0.05 using Mann-Whitney U test on the average number of completed tasks per session. Although Hta-Gre-Rel has the highest task throughput (1.8 tasks/min vs $\approx$ 1.6 tasks/min for other strategies), it has the lowest number of completed tasks since workers quickly dropped their work sessions when they were assigned tasks using Hta-Gre-Rel (average duration: 18.3 minutes). Also, the number of completed tasks for Hta-Gre grows steadily. This shows the superiority of Hta-Gre as it maintains workers' efficiency throughout their work session. At the very end, all curves decrease since workers need to submit their HIT before the allotted time expires (30 minutes).

We observed that workers who were assigned tasks using Hta-Gre completed 36.7 tasks per work session, with an average session length of 22.3 minutes and an average task reward of $0.064. Although workers who were assigned tasks using Random stayed slightly longer (22.7 minutes/session), they were less efficient (34 tasks/session) even though they chose tasks that pay slightly better (average task reward was $0.068). *So far, in addition to showing that* Hta-Gre *is superior in terms of number of completed tasks per session, we show that while* Hta-Gre-Div *provided the best outcome quality, it has the worst number of completed tasks. We conjecture that too much diversity results in overhead in choosing tasks. Similarly, providing relevant tasks only may induce boredom. A balance needs to be found with adaptability.*

**Worker retention.** Figure 3.12c shows worker retention as the percentage of work sessions (vertical axis) that ended after $x$ minutes (horizontal axis). This measure is akin to a *survival rate*, showing if workers are motivated to stay longer in a session. Random and Hta-Gre outperform Hta-Gre-Rel and Hta-Gre-Div. For instance, 85% of workers stayed over 18.2 minutes with Hta-Gre (and only 75% for Random). The fact that Random and Hta-Gre outperform Hta-Gre-Rel is confirmed at the significance level 0.1 using a Mann-Whitney U test on average session duration. Hta-Gre slightly outperforms Random for most of the time (up to 22 minutes exactly), while the average session duration is similar to Hta-Gre (Table 3.12d). We can therefore answer our initial question: Hta-Gre *offers the best compromise between crowdwork quality and overall performance in terms of number of completed tasks and worker retention.*

## 3.5 Ita and Hta Experiments: Discussion

In both Ita and Hta, we conducted experiments with real workers hired from AMT and we redirected them to our home-grown platform where they completed real tasks that we extracted from Crowdflower. In this section, we first compare the results obtained in our two sets of experiments. Then, we discuss our experimental setting and the challenges raised by conducting experiments on task assignment in crowdsourcing.

**Results.** First, our experiments show the need to enable *adaptability*. Our adaptive strategies DIV-PAY (Ita) and HTA-GRE (Hta) both offer good compromises on various performance dimensions when compared to other non-adaptive strategies. Additionally, in Ita, we observed that several workers had an evolving motivation while performing tasks: some of them clearly favored a motivation factor over another. This confirms the need for accounting for the evolution of motivation in task assignment.

Second, we observe that task relevance is important for maximizing the number of completed tasks, task throughput and worker retention. In Ita, task relevance is ensured in all strategies using an hard constraint on the tasks that could be assigned to a worker. The strategy RELEVANCE, that only optimizes task relevance, performed best on both task throughput and worker retention. In Hta, task relevance is only optimized in HTA-GRE and HTA-GRE-REL. The HTA-GRE strategy outperformed other strategies on number of completed tasks and worker retention. The number of completed tasks was lower for HTA-GRE-REL, but this strategy provided the best task throughput. This result is not surprising, as workers are naturally more efficient when completing tasks that best match their interest. We also observe that optimizing only diversity impacts negatively worker retention: DIVERSITY and HTA-GRE-DIV performed worse on this dimension. This suggests that workers do not like completing diverse tasks during a long time, without being motivated by another factor (payment or relevance).

We also observe surprising results: in Ita, DIVERSITY performs the worst on crowdwork quality while HTA-GRE-DIV is the best in Hta. Both DIVERSITY and HTA-GRE-DIV only optimize task diversity, hence we could expect similar results. We conjecture that it is an empirical effect and we also note that these strategies slightly differ: DIVERSITY enforces an hard constraint on task relevance to a worker while HTA-GRE-DIV is relevance-agnostic. Hence, we cannot carry out a direct comparison.

Finally, our experiments showed the efficiency of our algorithms. Although we did not report performance experiments in Ita, we verified the responsiveness of our algorithms: they can return tasks to a worker in less than a second when $|\mathcal{T}^i| = 5000$. We showed the efficiency of HTA-GRE in Hta in our performance experiments. Our online experiments confirmed the applicability of our algorithms in a web application with a real workload. This shows that we can enable motivation-aware task assignment under the constraints raised by an online context.

**Discussion.** Designing and conducting experiments on task assignment in crowdsourcing is not a straightforward task. We discuss the challenges raised by designing an experimental setting and their impact on our experiments.

First, the novelty of our approach required to design key components for implementing and evaluating our assignment strategies. We introduced various performance dimensions that are best relevant to an holistic approach and consider the objectives of both requesters and workers. Previous approaches on task assignment [47, 55, 135] mainly focused on requester-centric measures such as crowdwork quality or task throughput. Studies on workers' motivation also included worker retention [33]. We proposed to adopt these performance dimensions to evaluate task assignment. We believe they provide a comprehensive insight of a system's performance. Our work also required to define an assignment iteration. We performed an iteration every time $x$ tasks were completed (see Appendix A.2 for more details). We could also consider a time-based approach, that performs an assignment iteration every $m$ minutes. This problem is orthogonal to our work.

Second, crowdsourcing is still a maturing domain of investigation and we inherit some of its longstanding challenges. A number of studies [15, 82, 116] list challenges raised in crowdsourcing and demonstrate that designing an efficient crowdsourcing system is not a straightforward task. In our work, we designed and developed a home-grown platform to enable adaptive task assignment. In our experiments, we observed that most workers were comfortable with the proposed workflow. Some of them told us that they liked completing tasks on our platform (they sent an email). However, some workers did not follow instructions, dropped their work session without completing any task, worked longer than the allotted time or were affected by the way tasks are displayed (Section 3.3.3.2). To mitigate that, we ran additional experiments to obtain a coherent experimental setting. However, this shows the need to investigate how to design a proper crowdsourcing system and leverage the guidelines that were proposed in several previous studies [15, 82, 116]. This observation also raises the question of experiments reproducibility. On popular platforms such as AMT, the crowd is by nature volatile and the degree of commitment of workers varies [82] as well as their demographics and their skills. In this context, reproducing experimental results is difficult. Some of the surprising results that we observed support this idea. Therefore, our results should be interpreted as *insights* on the impact of various assignment strategies.

## 3.6 Related Work on Motivation-Aware Task Assignment

### Task Assignment in Crowdsourcing

Task assignment in crowdsourcing was largely studied. Previous studies include the design of *adaptive* algorithms, that focus on maximizing the quality of crowdwork [47, 67, 68, 135].

For instance, Fan et al. [47] leverage the similarity between tasks and the past answers of workers to design an adaptive algorithm that aims at maximizing the accuracy of crowdwork. Their work includes two main components. First, they estimate workers' accuracy by leveraging the similarity between tasks. The idea is that workers should perform similarly on similar tasks. Then, they design an adaptive task assignment framework that (i) identifies *top workers*, (ii) assigns tasks to top workers and (iii) assigns *test* tasks to other workers, whose accuracy is not precisely known or insufficient. Ho et al. [67] study an online setting, where the workers who are going to arrive on the platform have unknown skill. They design an algorithm where the skill level of sampled workers is learned and leveraged to assign all other workers to tasks. Zheng et al. [135] designed *QASCA*, an adaptive task assignment framework. They focus on optimizing crowdwork quality and incorporating directly two popular evaluation metrics in task assignment (accuracy, F-score). They tackle two natural challenges: the lack of ground truth for crowdsourced tasks and the need for an efficient assignment algorithms in an online context. They adopt a probabilistic model which require to adapt evaluation metrics and propose efficient assignment algorithms that are adapted to these metrics. None of these studies includes motivation factors in their model.

Other investigations focused on dynamically adjusting the task reward [48, 55] so as to satisfy a deadline or a budget constraint. They modeled the willingness of a worker to choose a task as a task acceptance probability featuring task reward as a parameter. These studies do not focus on task assignment as workers self-appoint themselves to tasks and they do not include task diversity in their model. Recently, Rahman et al. [106] focused on assigning tasks to groups of *diverse workers* in collaborative crowdsourcing. Moreover, Wu et al. aim at finding sets of workers with diverse *opinions* [130]. None of those studies assigns *diverse tasks* to workers or includes motivation factors. Moreover, Rahman et al. [106] consider a setting which is not *adaptive*: task assignment does not leverage previous answers to improve the main objective.

### Motivating Workers

A range of studies point out the importance of suitably motivating workers in crowdsourcing [15, 82, 90]. Obviously, reward is an important factor, and crowdsourcing platforms should follow some guidelines that would solve wage issues [15]. Kittur et al. [82] underlines the interest of designing frameworks that include incentive schemes other than financial ones. In particular, they notice that a system should "*achieve both effective task completion and worker satisfaction*". Worker motivation was first studied in physical workplaces [62]. Recent studies [79] investigated the importance of 13 motivation factors for workers on Amazon Mechanical Turk. Although task payment remains the most important factor, Kaufmann et al. [79] point out that workers are also interested in *skill variety* or *task autonomy.*

Some efforts were driven towards experimenting motivation factors in crowdsourcing [25, 33, 108, 114]. In a recent study [33], Dai et al. inserted *diversions* in the workflow such that workers were presented with some entertainment contents between two task completions. Dai et al. showed that such a motivational scheme improved worker retention. Chandler and Kapelner [25] conducted experiments showing that workers perceiving the "meaningfulness" of task improved throughput without degrading quality. Another study [108] assessed the effect of extrinsic and intrinsic motivation factors. They demonstrated that workers were more accurate on meaningful tasks posted by a non-profit organization than on tasks posted by a private firm and less explicit about their outcome. This suggested that intrinsic factors help improve quality of crowdwork. Shaw et al. [114] assessed 14 incentives schemes and found that incentives based on worker-to-worker comparisons yield better crowdwork quality. None of the above studies leverages motivation factors to optimize task assignment, and thus they do not tackle the motivation-aware task assignment problem. Hata et al. [66] studied how work quality changes over extended periods of time.

#### Diversity Formulations and Algorithms

Diversity is a widely studied subject that finds its roots in Web search with a goal similar to ours. For example, it was used in text retrieval and summarization to balance document relevance and novelty [22]. Most approaches fall into two cases: *content-based* [22] and *intent-based* [28]. Gollapudi and Sharma [59] adopt an axiomatic approach to diversity to address user intent. They show that no diversity function can satisfy all axioms together. Our formulation is content-based as it relies on a distance function between tasks. Other content-based functions, such as ones based on taxonomies, are possible [8]. In our work, it is essential to use a metric to obtain our approximation results.

In the database context, Chen and Li [98] propose to post-process structured query results to enforce diversity. Similarly, in recommendations [42, 137], intermediate results are post-processed, using pairwise item similarity, to generate accurate and diverse recommendations. Vee et al. [122] introduced a hierarchical notion of diversity in databases and proposed efficient top-k processing algorithms. Abbar et al. [3] proposed an algorithm with provable approximation guarantees to find relevant and diverse news articles.

## 3.7 Conclusion

In this chapter, we investigated monitoring in crowdsourcing. We advocated the need to incorporate motivation in task assignment. Our approach is adaptive as it relies on observing workers in task completion, capturing their motivation and using it to determine the next tasks to assign to them.

We modeled three motivation factors: task diversity, task payment and task relevance. Then, we studied two variants of task assignment where we define a worker's expected motivation using two combinations of these factors.

First, we investigated *Individual Task Assignment* (Ita) [101], where we assign tasks to workers individually, one worker at a time. In this variant, we defined motivation as a balance between task diversity and task payment. We modeled Ita and showed it is NP-Hard. We designed three task assignment strategies that exploit various objectives: RELEVANCE that assigns tasks matching a worker's profile, DIVERSITY that chooses matching and diverse tasks and DIV-PAY an adaptive strategy that selects matching tasks with the best compromise between diversity and payment. In practice, our experiments showed that different strategies prevail. RELEVANCE offers the best task throughput and worker retention since it requires less context switching for workers. DIV-PAY, however, has the best outcome quality, since it accounts for workers' motivation by allowing them to achieve the best compromise between diversity and compensation. This confirms the need for adaptive motivation-aware task assignment.

Second, we studied *Holistic Task Assignment* (Hta) [102], where we assign tasks to all available workers, holistically. We defined motivation as a combination of task diversity and task relevance. We modeled Hta and showed it is both NP-Hard and MaxSNP-Hard. We developed HTA-APP and HTA-GRE, two approximation algorithms for Hta, that have $\frac{1}{4}$ and $\frac{1}{8}$ approximation factors, respectively. Our offline experiments examined the response time and the value of the objective function for both algorithms and showed that HTA-GRE is faster than HTA-APP, without compromising motivation (i.e. the value of the objective function). We also deployed online experiments with real workers and compared HTA-GRE with various non-adaptive assignment strategies. We showed that optimizing diversity only results in the highest crowdwork quality and optimizing relevance only is worst. We also found that HTA-GRE offers the best compromise between those dimensions thereby assessing the need for adaptability.

# Chapter 4

# Summary and Perspectives

In this thesis, we developed algorithms for monitoring activity traces. Section 4.1 concludes our work and Section 4.2 discusses some perspectives and future investigations.

## 4.1 Summary

In Chapter 1, we showed the importance of designing scalable algorithms for monitoring activity traces. We introduced two kinds of activity traces that are of interest to us: temporal data and traces of human activity in crowdsourcing. For temporal data, we discussed the need to study temporal joins featuring an approximate interpretation of temporal predicates. In crowdsourcing, we advocated the need to incorporate motivation in adaptive task assignment to account for the evolution of workers' motivation and improve overall performance.

In Chapter 2, we studied the efficient evaluation of temporal joins using two different processing paradigms. In Section 2.3, we investigated batch processing and formalized Ranked Temporal Join (RTJ), that are often best interpreted as top-$k$ queries where only the best matches are returned. In Section 2.3.2, we showed how to exploit the nature of temporal predicates and the properties of their associated scoring semantics to design *TKIJ*, an efficient RTJ query evaluation approach on a distributed Map-Reduce architecture. In Section 2.3.3, we presented extensive experiments conducted on synthetic and real datasets showing that *TKIJ* outperforms state-of-the-art competitors and provides very good performance for n-ary RTJ queries on temporal data.

We proposed a preliminary study for extending our work to stream processing (Section 2.4). We revisited the RTJ model and outlined several directions to design an efficient distributed query evaluation approach. We focused on tackling challenges raised by distributed processing and by the need for a low-latency system.

In Chapter 3, we studied a motivation-aware task assignment approach in crowdsourcing. First, we introduced and modeled three motivation factors: task diversity, task payment and task relevance (Section 3.2). Then, we studied two variants of task assignment: Individual Task Assignment (Ita) and Holistic Task Assignment (Hta).

First, we modeled Ita, where we define motivation as a balance between task diversity and task payment (Section 3.3). We showed that Ita is NP-Hard (Theorem 1). We proposed three assignment strategies for Ita: RELEVANCE, DIV-PAY and DIVERSITY. We conducted experiments (Section 3.3.3) with real workers showing that DIV-PAY, an adaptive motivation-aware approach that optimizes both task diversity and task payment leads to higher quality contribution. RELEVANCE, a motivation-agnostic strategy that assigns tasks matching a worker's profile, leads to higher task throughput and worker retention.

Second, we modeled Hta (Section 3.4), where motivation is defined as a combination of task diversity and task relevance. We show that Hta is both NP-Hard and MaxSNP-Hard (Theorems 2 and 3). We studied two approximation algorithms for Hta: HTA-APP and HTA-GRE. The algorithm HTA-APP has a better approximation factor with the cost of a higher running time (Section 3.4.2). We conducted experiments on synthetic data showing that HTA-GRE outperforms HTA-APP (Section 3.4.3). Then, we conducted experiments with real workers and compared HTA-GRE with various non-adaptive strategies. We showed that HTA-GRE offers the best compromise between performance dimensions.

In both Ita and Hta, we conducted live experiments using a home-grown platform (GACS) to enable adaptive task assignment (Appendix A). We recruited workers from Amazon Mechanical Turk and published real tasks released by Crowdflower.

## 4.2 Perspectives

We envision three main perspectives for future work. The first is immediate future work and aims to improve the way we tackle problems related to our main line of investigation. We also outline future work that require deeper investigations: we could explore monitoring using other platforms and extend our study using wider semantics.

### 4.2.1 Optimizations

In our two research areas, we identified possible improvements for solving problems that are related to our main line of investigation.

**Data Partitioning.** Our work on temporal data monitoring aims at scaling to large datasets by leveraging parallel processing in distributed systems. We believe that partitioning is a natural solution to enable parallel join processing on a set of small independent tasks that can be executed simultaneously. In the context of temporal joins, time-based partitioning is acknowledged as an effective method [29, 35]. In our work, we defined the granularity of partitioning *empirically*, by finding a sweet spot for the number of granules $g$. One possible improvement is to derive *analytically* or *algorithmically* the optimal value for $g$ using a cost-based approach. Previous work [35] has investigated how to derive the optimal number of granules for "intersection" joins[1]. We could adapt these techniques to our setting to find the optimal value for $g$.

**Capturing Motivation.** We could investigate various ways to capture a worker's motivation at iteration $i$. In our work, we leveraged a collection of observations, made each time a worker completes a task. For each completed task, we simply computed the marginal gain in a given factor (Section 3.2) and aggregated these measures to compute $\alpha_w^i$ or $\beta_w^i$.

First, we could ask directly workers their preference for the given factors. This has the advantage of simplicity and ensures accuracy. However, workers may not be able to identify what actually motivates them and they may prefer completing tasks rather than answering a question they are not compensated for. We could use other techniques for capturing automatically $\alpha_w^i$ and $\beta_w^i$. For instance, we could map the completed tasks in a vector space and use the centroid of these tasks to obtain worker's preferences. Suppose that a worker $w$ completed tasks $\{t_1, \ldots, t_J\}$ where $\forall j \in \{1, \ldots, J\}\, t_j \in \mathcal{T}_w^{i-1}$ during iteration iteration $i-1$. Suppose that we adopt the $\mathsf{Hta}$ variant of motivation-aware task assignment, where the parameter $\alpha_w^i$ (resp. $\beta_w^i$) captures the preference of $w$ for task diversity (resp. task relevance). We adopt a vector space with one dimension for each factor preference. We represent each task $t_k \in \mathcal{T}_w^{i-1}$ using a vector $\mathbf{u_{t_k}} = \langle rel(w, t_k), d_{t_k} \rangle$. Here, $d_{t_k} = \sum_{t_l \in \{t_1, \ldots t_J\}} d(t_k, t_l)$ refers to the gain in task diversity brought by $t_k$. Then, we compute the centroid of all completed tasks and we use its coordinates to compute $\alpha_w^i$ and $\beta_w^i$ (some normalization will be necessary).

Another possible investigation is to leverage the *sequence* of task completed by a worker. We may consider that the *order* and the characteristics of tasks in this sequence characterizes which factor best motivates the worker during iteration $i$. We could leverage Hidden Markov Models (HMM) to model a sequence of completed tasks. Indeed, a sequence of completed tasks can be seen as a sequence of *observations*. Each *state* could represent a motivation factor, that leads a worker to choose a specific task during iteration $i$.

Suppose now that we have a set of HMM, where each HMM is associated to a worker that would have a given compromise between motivation factors. For instance, one HMM may be associated to a worker who is equally motivated by task diversity and task relevance. This HMM models the sequence of tasks that such a worker is likely to complete. Given a sequence of observations (i.e. a sequence of completed tasks), we could find the HMM that generates this sequence with the highest probability. This allows to classify each worker based on the tasks that she completed and capture her compromise between motivation factors. However, we still need to *learn* each HMM that is associated to a given compromise between motivation factors. To do so, we could collect training data by observing workers completing tasks and asking them their preferences between motivation factors (i.e. their profile). The sequence of completed tasks and the collected profiles are the examples used

[1]Also known as "overlap" joins.

to train each HMM. Then, we each profile is associated to a HMM that is learned using the Baum-Welch algorithm [31].

We could compare these approaches to discover which one best matches workers' motivation and study their impact on performance dimensions such as task throughput.

### 4.2.2 Platforms

Our investigations on temporal data monitoring focused on optimizing the evaluation of a new kind of query (RTJ). We aimed at designing a distributed query evaluation approach, and our main contribution is *TKIJ*, an efficient RTJ query evaluation approach on a Map-Reduce architecture. We present future investigation for evaluating RTJ queries using other distributed platforms.

**Leveraging Communication Between Workers.** In *TKIJ*, reducers do not communicate between them. Broadcasting the score of the $k$-th result would allow to terminate earlier join processing. Indeed, each reducer evaluates locally a full RTJ query, which is akin to a top-$k$ query. During top-$k$ processing, it is natural to compare the score of the current $k$-th result to the maximum possible score of results that are not yet evaluated : if the $k$-th result has a high score, we can terminate top-$k$ processing and return results. Therefore, communicating the current $k$-th score would allow to terminate earlier the reduce phase in *TKIJ*. However, since Map-Reduce does not support natively communication between reducers, the use of other distributed platforms would be necessary.

**Stream Processing.** Evaluating RTJ queries in the context of stream processing is a natural extension of our work on batch processing (*TKIJ*). We propose a detailed preliminary study in Section 2.4.

### 4.2.3 Semantics

In our two research areas, we studied new problems that were not previously investigated. We formalized these problems and, as a first attempt, we focused on supporting specific semantics that are of interest to us. We identified three main research areas where we can extend our investigation using wider semantics.

**Supporting Hybrid Queries.** In temporal data monitoring, we proposed to study a new kind of query coined RTJ. RTJ queries feature predicates that only compare interval *endpoints*. We could support richer semantics by including *other interval attributes* in the query. For instance, in network traffic monitoring, a connection is characterized by its endpoints but also by the IP addresses of the client and of the server. An analyst could be interested in obtaining pairs of connections that have the same client IP address. Therefore, she would need to formulate *equi-join* queries, that use the client IP address in the join condition. Our queries would be *hybrid* and feature RTJ semantics and other join conditions, such as equality. These queries were studied in previous work on interval joins using Map-Reduce [29], that only support a Boolean interpretation of Allen predicates.

A key challenge for evaluating such queries is to *efficiently* return *correct* results. In *TKIJ*, we rely on statistics' collection and on the *TopBuckets* process to prune only unnecessary results. We would need to adapt these components to handle hybrid queries. For instance, for equi-joins, we could use histograms to record, in each histogram bucket, the number of intervals that correspond to a distinct value of a given attribute. Then, we could use these statistics to prune unnecessary results. One pitfall of this approach is that an attribute may have a high cardinality domain, which would cause an overhead [37]. To tackle this problem, we could use approximate statistics [37]. However we aim at returning *exact* top-$k$ results. We could also leverage Bloom filters, that allow to have a compact structure storing interval attributes [95]. However, Bloom filters incur false positives, which would require several processing iterations, causing an overhead in our Map-Reduce context.

**Collaborative Tasks.** In our work on monitoring human activity in crowdsourcing, we focused on micro-tasks that are completed independently by workers. Yet, a number of tasks require collaboration. For instance, writing a news summary or a Wikipedia article requires collaboration and organization between workers. Other examples include generation of subtitles[2] [106] or collaborative maps (OpenStreetMap, CrowdMap).

We could extend our investigations on motivation-aware task assignment to collaborative tasks. In this context, task assignment would need to account for the presence of several workers in forming the most motivated *team* to complete a task.

Forming motivated teams require to devise a proper model. We need to define a team's motivation, that can be seen as a combination of individual and *collective* motivation. Previous work [79] identified *community identification* and *social contact* as *community-based* motivation factors. Although these factors are not *team-based*, they show the importance of relations between workers in motivation. In collaborative task assignment, Rahman et al. [106] modeled *affinity* between workers to find the most *efficient* team to complete a task. Here, affinity is seen as similarity between workers using socio-demographic attributes such as gender, age or region. For each task, they aim at finding a group that minimizes its affinity diameter. They also propose to split that group into sub-groups if its *critical mass* is exceeded. Rahman et al. also propose approximation algorithms for their problem which is proven to be NP-Hard. This shows that motivation-aware collaborative task assignment would probably require to design efficient algorithms to solve hard problems: this is our second challenge.

To the best of our knowledge, no previous study has investigated motivation-aware task assignment for collaborative tasks. The work of Rahman and al. [79] is the closest to this

[2]Also known as "Fan-subbing" `https://en.wikipedia.org/w/index.php?title=Fansub&oldid=766250166`.

problem. However, they do not include explicitly motivation in their model and they do not study *adaptive* task assignment: they do not leverage previous answers.

**Motivation Model.** In our investigation on monitoring in crowdsourcing, we defined the expected motivation of a worker as a combination of three main motivation factors: *task diversity*, *task payment* and *task relevance*. We could investigate the impact of other motivation factors and adopt a different model.

First, we could study other factors that were identified by Kaufman et al. [79]. For instance, *task autonomy* refers to "the degree of freedom that a worker is allowed during task execution". Task autonomy is not relevant to micro-tasks that do not allow a high degree of freedom. However, it could be interesting to study in the context of creative tasks (e.g. writing an article). Other examples include *human capital advancement*, that captures the possibility of a worker to learn or perfect a skill when completing a task. This may be the case for tasks that require specific knowledge (e.g. text translation task).

Another model developed for commerce search [84] is leveraged in task assignment [55] to capture the willingness of a worker to complete a task. Specifically, Gao et al [55] modeled the *task acceptance probability* that captures how prone a worker is to choose a task. They focus on adjusting task reward so as to meet deadlines, hence task acceptance only depends on task reward. We could revisit the original model proposed by Li et al. [84] and include other task characteristics.

# Bibliography

[1] Crowdflower - data for everyone. `https://www.crowdflower.com/data-for-everyone/`.

[2] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. The design of the borealis stream processing engine. In *CIDR*, pages 277–289, 2005.

[3] Sofiane Abbar et al. Real-time recommendation of diverse related articles. In *WWW*, pages 1–12, 2013.

[4] Foto N. Afrati and Jeffrey D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010.

[5] Foto N. Afrati and Jeffrey D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.

[6] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In *SIGMOD*, pages 147–160, 2008.

[7] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[8] Aris Anagnostopoulos et al. Sampling search-engine results. *WWW*, 2006.

[9] Rajagopal Ananthanarayanan, Venkatesh Basker, Sumit Das, Ashish Gupta, Haifeng Jiang, Tianhao Qiu, Alexey Reznichenko, Deomid Ryabkov, Manpreet Singh, and Shivakumar Venkataraman. Photon: fault-tolerant and scalable joining of continuous data streams. In *SIGMOD*, pages 577–588, 2013.

[10] Esther M. Arkin et al. Approximating the maximum quadratic assignment problem. *Inf. Process. Lett.*, pages 13–16, 2001.

[11] Giorgio Ausiello et al. Approximation preserving reductions. In *Complexity and Approximation*. Springer, 1999.

[12] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS*, pages 1–16, 2002.

[13] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *SIGMOD*, pages 28–39. ACM, 2003.

[14] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[15] Benjamin B. Bederson and Alexander J. Quinn. Web workers unite! addressing challenges of online laborers. In *CHI*, pages 97–106, 2011.

[16] AS Besicovitch. On a general metric property of summable functions. *Journal of the London Mathematical Society*, 1(2):120–128, 1926.

[17] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, and Yuanyuan Tian. A comparison of join algorithms for log processing in mapreduce. In *SIGMOD*, pages 975–986, 2010.

[18] Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *PODS*, pages 155–166, 2012.

[19] Rainer E. Burkard et al. *Assignment Problems*. SIAM, 2009.

[20] K. Selçuk Candan, Jong Wook Kim, Parth Nagarkar, Mithila Nagendra, and Renwei Yu. Rankloud: Scalable multimedia data processing in server clusters. *IEEE MultiMedia*, 18(1):64–77, 2011.

[21] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink$^{\text{TM}}$: Stream and batch processing in a single engine. *IEEE Data Eng. Bull.*, 38(4):28–38, 2015.

[22] J. G. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Research and Development in Information Retrieval*, 1998.

[23] Michael J. Carey and Donald Kossmann. On saying "enough already!" in SQL. In *SIGMOD*, pages 219–230, 1997.

[24] Giorgio Carpaneto et al. Algorithms and codes for the assignment problem. *Annals of operations research*, pages 191–223, 1988.

[25] Dana Chandler and Adam Kapelner. Breaking monotony with meaning: Motivation in crowdsourcing markets. *CoRR*, abs/1210.0962, 2012.

[26] Barun Chandra and Magnús M. Halldórsson. Approximation algorithms for dispersion problems. *J. Algorithms*, 38(2):438–465, 2001.

[27] Kevin Chen-Chuan Chang and Seung-won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *SIGMOD*, pages 346–357, 2002.

[28] Olivier Chapelle et al. Intent-based diversification of web search results: metrics and algorithms. *Information Retrieval*, pages 572–592, 2011.

[29] Bhupesh Chawda, Himanshu Gupta, Sumit Negi, Tanveer A. Faruquie, L. Venkata Subramaniam, and Mukesh K. Mohania. Processing interval joins on map-reduce. In *EDBT*, pages 463–474, 2014.

[30] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *MONET*, 19(2):171–209, 2014.

[31] Antoine Cornuéjols and Laurent Miclet. *Apprentissage artificiel: concepts et algorithmes*. Editions Eyrolles, 2011.

[32] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.

[33] Peng Dai, Jeffrey M. Rzeszotarski, Praveen Paritosh, and Ed H. Chi. And now for something completely different: Improving crowdsourcing workflows with micro-diversions. In *ACM CSCW*, pages 628–638, 2015.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.

[35] Anton Dignös, Michael H. Böhlen, and Johann Gamper. Overlap interval partition join. In *SIGMOD*, pages 1459–1470, 2014.

[36] Christos Doulkeridis and Kjetil Nørvåg. On saying "enough already!" in mapreduce. In *Cloud-I*, 2012.

[37] Christos Doulkeridis, Akrivi Vlachou, Kjetil Nørvåg, Yannis Kotidis, and Neoklis Polyzotis. Processing of rank joins in highly distributed systems. In *ICDE*, pages 606–617, 2012.

[38] Doratha E. Drake and Stefan Hougardy. A simple approximation algorithm for the weighted matching problem. *Inf. Process. Lett.*, pages 211–213, 2003.

[39] Maciej Drozdowski. *Scheduling for Parallel Processing*. Computer Communications and Networks. Springer, 2009.

[40] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 2014.

[41] Didier Dubois, Allel HadjAli, and Henri Prade. Fuzziness and uncertainty in temporal reasoning. *J. UCS*, 9(9):1168, 2003.

[42] Khalid El-Arini et al. Turning down the noise in the blogosphere. In *SIGKDD*, pages 289–298, 2009.

[43] Mohammed Elseidy, Abdallah Elguindy, Aleksandar Vitorovic, and Christoph Koch. Scalable and adaptive online joins. *PVLDB*, 7(6):441–452, 2014.

[44] Jost Enderle, Matthias Hampel, and Thomas Seidl. Joining interval data in relational databases. In *SIGMOD*, pages 683–694, 2004.

[45] Ronald Fagin. Combining fuzzy information from multiple systems. In *PODS*, pages 216–226, 1996.

[46] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS*, pages 102–113, 2001.

[47] Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. icrowd: An adaptive crowdsourcing framework. In *SIGMOD*, pages 1015–1030, 2015.

[48] Siamak Faradani, Bjoern Hartmann, and Panagiotis G. Ipeirotis. What's the right price? pricing tasks for finishing on time. In *AAAI*, 2011.

[49] Sándor P. Fekete and Henk Meijer. Maximum dispersion and geometric maximum weight cliques. *CoRR*, cs.DS/0310037, 2003.

[50] Thomas A. Feo and Mallek Khellaf. A class of bounded approximation algorithms for graph partitioning. *Networks*, pages 181–195, 1990.

[51] Jonathan Finger and Neoklis Polyzotis. Robust and efficient algorithms for rank join evaluation. In *SIGMOD*, pages 415–428, 2009.

[52] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[53] Ioannis Flouris, Nikos Giatrakos, Antonios Deligiannakis, Minos N. Garofalakis, Michael Kamp, and Michael Mock. Issues in complex event processing: Status and prospects in the big data era. *Journal of Systems and Software*, 127:217–236, 2017.

[54] Dengfeng Gao, Christian S. Jensen, Richard T. Snodgrass, and Michael D. Soo. Join operations in temporal databases. *VLDB J.*, 14(1):2–29, 2005.

[55] Yihan Gao and Aditya G. Parameswaran. Finish them!: Pricing algorithms for human computation. *PVLDB*, 7(14):1965–1976, 2014.

[56] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[57] Lukasz Golab and M. Tamer Özsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.

[58] Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *VLDB*, pages 500–511, 2003.

[59] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[60] Xiaohui Gu, Philip S. Yu, and Haixun Wang. Adaptive load diffusion for multiway windowed stream joins. In *ICDE*, pages 146–155, 2007.

[61] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[62] J. Hackman and G. R. Oldham. Motivation through the design of work: Test of a theory. *Organizational Behavior and Human Performance*, 16(22):250–279, 1976.

[63] Moustafa A. Hammad, Walid G. Aref, and Ahmed K. Elmagarmid. Query processing of multi-way stream window joins. *VLDB J.*, 17(3):469–488, 2008.

[64] Refael Hassin and Shlomi Rubinstein. An improved approximation algorithm for the metric maximum clustering problem with given cluster sizes. *Inf. Process. Lett.*, 98(3):92–95, 2006.

[65] Refael Hassin, Shlomi Rubinstein, and Arie Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3):133–137, 1997.

[66] Kenji Hata et al. A glimpse far into the future: Understanding long-term crowd worker quality. In *CSCW*, 2017.

[67] Chien-Ju Ho, Shahin Jabbari, and Jennifer Wortman Vaughan. Adaptive task assignment for crowdsourced classification. In *ICML*, pages 534–542, 2013.

[68] Chien-Ju Ho and Jennifer Wortman Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, 2012.

[69] John Joseph Horton and Lydia B. Chilton. The labor economics of paid crowdsourcing. In *ACM EC*, pages 209–218, 2010.

[70] Jeff Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.

[71] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Joining ranked inputs in practice. In *VLDB*, pages 950–961, 2002.

[72] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, pages 754–765, 2003.

[73] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-$k$ query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[74] Panagiotis G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *ACM Crossroads*, 17(2):16–21, 2010.

[75] Adam Jacobs. The pathologies of big data. *Commun. ACM*, 52(8):36–44, 2009.

[76] Yuanzhen Ji, Jun Sun, Anisoara Nica, Zbigniew Jerzak, Gregor Hackenbroich, and Christof Fetzer. Quality-driven disorder handling for m-way sliding window stream joins. In *ICDE*, pages 493–504, 2016.

[77] Jaewoo Kang, Jeffrey F. Naughton, and Stratis Viglas. Evaluating window joins over unbounded streams. In *ICDE*, pages 341–352, 2003.

[78] Martin Kaufmann, Amin Amiri Manjili, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Färber, and Norman May. Timeline index: a unified data structure for processing queries on temporal data in SAP HANA. In *SIGMOD*, pages 1173–1184, 2013.

[79] Nicolas Kaufmann, Thimo Schulze, and Daniel Veit. More than fun and money. worker motivation in crowdsourcing - A study on mechanical turk. In *AMCIS*, 2011.

[80] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems.* Springer, 2004.

[81] Younghoon Kim and Kyuseok Shim. Parallel top-k similarity join algorithms using mapreduce. In *ICDE*, pages 510–521, 2012.

[82] Aniket Kittur, Jeffrey V. Nickerson, Michael S. Bernstein, Elizabeth Gerber, Aaron D. Shaw, John Zimmerman, Matt Lease, and John Horton. The future of crowd work. In *CSCW*, pages 1301–1318, 2013.

[83] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M. Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter heron: Stream processing at scale. In *SIGMOD*, pages 239–250, 2015.

[84] Beibei Li, Anindya Ghose, and Panagiotis G. Ipeirotis. Towards a theory model for product search. In *WWW*, pages 327–336, 2011.

[85] Feifei Li, Ke Yi, and Wangchao Le. Top-$k$ queries on temporal data. *VLDB J.*, 19(5):715–733, 2010.

[86] Ming Li, Murali Mani, Elke A. Rundensteiner, and Tao Lin. Complex event pattern detection over streams with interval-based temporal semantics. In *DEBS*, pages 291–302, 2011.

[87] Qian Lin, Beng Chin Ooi, Zhengkui Wang, and Cui Yu. Scalable distributed stream join processing. In *SIGMOD*, pages 811–825, 2015.

[88] Hongjun Lu, Beng Chin Ooi, and Kian-Lee Tan. On spatially partitioned temporal join. In *VLDB*, pages 546–557, 1994.

[89] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *PVLDB*, 5(10):1016–1027, 2012.

[90] David B. Martin, Benjamin V. Hanrahan, Jacki O'Neill, and Neha Gupta. Being a turker. In *CSCW*, pages 224–235, 2014.

[91] Gianmarco De Francisci Morales and Aristides Gionis. Streaming similarity self-join. *PVLDB*, 9(10):792–803, 2016.

[92] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, pages 635–646, 2006.

[93] Viswanath Nagarajan and Maxim Sviridenko. On the maximum quadratic assignment problem. In *SODA*, pages 516–524, 2009.

[94] Apostol Natsev, Yuan-Chi Chang, John R. Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. Supporting incremental join queries on ranked inputs. In *VLDB*, pages 281–290, 2001.

[95] Nikos Ntarmos, Ioannis Patlakas, and Peter Triantafillou. Rank join queries in nosql databases. *PVLDB*, 7(7):493–504, 2014.

[96] Hans Jürgen Ohlbach. Relations between fuzzy time intervals. In *TIME*, pages 44–51, 2004.

[97] Alper Okcan and Mirek Riedewald. Processing theta-joins using mapreduce. In *SIGMOD*, pages 949–960, 2011.

[98] Zhiyuan Chen others. Addressing diverse user preferences in sql-query-result navigation. In *SIGMOD*, 2007.

[99] Wenceslao Palma, Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Dhtjoin: processing continuous join queries using DHT networks. *Distributed and Parallel Databases*, 26(2-3):291–317, 2009.

[100] Danila Piatov, Sven Helmer, and Anton Dignös. An interval join optimized for modern hardware. In *ICDE*, pages 1098–1109, 2016.

[101] Julien Pilourdault, Sihem Amer-Yahia, Dongwon Lee, and Senjuti Basu Roy. Motivation-aware task assignment in crowdsourcing. In *EDBT*, pages 246–257, 2017.

[102] Julien Pilourdault, Sihem Amer-Yahia, Senjuti Basu Roy, and Dongwon Lee. Holistic motivation-aware task assignment in crowdsourcing. Under Review.

[103] Julien Pilourdault, Vincent Leroy, and Sihem Amer-Yahia. Distributed evaluation of top-k temporal joins. In *SIGMOD*, pages 1027–1039, 2016.

[104] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco3 Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2014.

[105] Zhengping Qian, Yong He, Chunzhi Su, Zhuojie Wu, Hongyu Zhu, Taizhi Zhang, Lidong Zhou, Yuan Yu, and Zheng Zhang. Timestream: reliable stream computation in the cloud. In *EuroSys*, pages 1–14, 2013.

[106] Habibur Rahman, Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Task assignment optimization in collaborative crowdsourcing. In *IEEE ICDM*, pages 949–954, 2015.

[107] Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri Kumar Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[108] Jakob Rogstadius, Vassilis Kostakos, Aniket Kittur, Boris Smus, Jim Laredo, and Maja Vukovic. An assessment of intrinsic and extrinsic motivation on task performance in crowdsourcing markets. In *ICWSM*, 2011.

[109] Senjuti Basu Roy, Ioanna Lykourentzou, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Task assignment optimization in knowledge-intensive crowdsourcing. *VLDB J.*, 24(4):467–491, 2015.

[110] Karl Schnaitter and Neoklis Polyzotis. Evaluating rank joins with optimal cost. In *PODS*, pages 43–52, 2008.

[111] Steven Schockaert, Martine De Cock, and Etienne E. Kerre. Fuzzifying allen's temporal interval relations. *IEEE T. Fuzzy Systems*, 16(2):517–533, 2008.

[112] Nicholas Poul Schultz-Møller, Matteo Migliavacca, and Peter R. Pietzuch. Distributed complex event processing with query rewriting. In *DEBS*, 2009.

[113] George Sfakianakis, Ioannis Patlakas, Nikos Ntarmos, and Peter Triantafillou. Interval indexing and querying on key-value cloud stores. In *ICDE*, pages 805–816, 2013.

[114] Aaron D. Shaw, John J. Horton, and Daniel L. Chen. Designing incentives for inexpert human raters. In *CSCW*, pages 275–284, 2011.

[115] Zhitao Shen, Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Haixun Wang. A generic framework for top-k pairs and top-k objects queries over sliding windows. *TKDE*, 26(6):1349–1366, 2014.

[116] Aleksandrs Slivkins and Jennifer Wortman Vaughan. Online decision making in crowdsourcing markets: theoretical challenges. *SIGecom Exchanges*, 12(2):4–23, 2013.

[117] Richard T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

[118] Michael Stonebraker, Daniel J. Abadi, David J. DeWitt, Samuel Madden, Erik Paulson, Andrew Pavlo, and Alexander Rasin. Mapreduce and parallel dbmss: friends or foes? *Commun. ACM*, 53(1):64–71, 2010.

[119] Michael Stonebraker, Ugur Çetintemel, and Stanley B. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.

[120] Jens Teubner and René Müller. How soccer players would do stream joins. In *SIGMOD*, pages 625–636, 2011.

[121] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthikeyan Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy V. Ryaboy. Storm@twitter. In *SIGMOD*, pages 147–156, 2014.

[122] Erik Vee et al. Efficient computation of diverse query results. In *ICDE*, pages 228–236, 2008.

[123] Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *VLDB*, pages 285–296, 2003.

[124] Aleksandar Vitorovic, Mohammed Elseidy, Khayyam Guliyev, Khue Vu Minh, Daniel Espino, Mohammad Dashti, Yannis Klonatos, and Christoph Koch. Squall: Scalable real-time analytics. *PVLDB*, 9(13):1553–1556, 2016.

[125] Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. Angle-based space partitioning for efficient parallel skyline computation. In *SIGMOD*, pages 227–238, 2008.

[126] Akrivi Vlachou, Christos Doulkeridis, Kjetil Nørvåg, and Michalis Vazirgiannis. On efficient top-k query processing in highly distributed environments. In *SIGMOD*, pages 753–764, 2008.

[127] Song Wang and Elke A. Rundensteiner. Scalable stream join processing with expensive predicates: workload distribution and adaptation by time-slicing. In *EDBT*, pages 299–310, 2009.

[128] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *SIGMOD*, pages 407–418, 2006.

[129] Minji Wu, Jianliang Xu, Xueyan Tang, and Wang-Chien Lee. Top-k monitoring in wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(7), 2007.

[130] Ting Wu, Lei Chen, Pan Hui, Chen Jason Zhang, and Weikai Li. Hear the whole story: Towards the diversity of opinion in crowdsourcing markets. *PVLDB*, 8(5):485–496, 2015.

[131] Matei Zaharia, Tathagata Das, Haoyuan Li, Scott Shenker, and Ion Stoica. Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters. In *USENIX*, 2012.

[132] Donghui Zhang, Vassilis J. Tsotras, and Bernhard Seeger. Efficient temporal join processing using indices. In *ICDE*, pages 103–113, 2002.

[133] Haopeng Zhang, Yanlei Diao, and Neil Immerman. Recognizing patterns in streams with imprecise timestamps. *PVLDB*, 3(1):244–255, 2010.

[134] Xiaofei Zhang, Lei Chen, and Min Wang. Efficient multi-way theta-join processing using mapreduce. *PVLDB*, 5(11):1184–1195, 2012.

[135] Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. QASCA: A quality-aware task assignment system for crowdsourcing applications. In *SIGMOD*, pages 1031–1046, 2015.

[136] Yongluan Zhou, Ying Yan, Feng Yu, and Aoying Zhou. Pmjoin: Optimizing distributed multi-way stream joins by stream partitioning. In *DASFAA*, pages 325–341, 2006.

[137] Cai-Nicolas Ziegler et al. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.

# Appendix A

# Motivation-Aware Crowdsourcing: Experimental Setup

This appendix presents components of our experiments in crowdsourcing. First, we present tasks datasets that are used in both synthetic and live experiments to evaluate our algorithms and assignment strategies. Second, we developed a platform to enable adaptive crowdsourcing and motivation-aware task assignment. We present the workflow on this platform and how workers are recruited.

## A.1 Tasks Datasets

**Tasks for Live Experiments** We employ a first set of tasks that we publish on our platform: workers are paid to complete these tasks. We use a set of 158,018 micro-tasks released by Crowdflower [1]. It includes 22 different kinds of tasks, featuring tweet classification, searching information on the web, transcription of images, sentiment analysis, entity resolution or extracting information from news. Each different kind of task is assigned a set of keywords that best describe its content and a reward, ranging from \$0.01 to \$0.12. Considered tasks are *micro-tasks* (they take from 5s to 60s to be completed, depending on how well a worker knows its instructions). We set reward proportional to the expected completion time.

**Task for Synthetic Experiments** We use real tasks for experiments with synthetic workers. We crawled 152,221 *task groups* from Amazon Mechanical Turk (AMT). Each task group contains id, title, reward, description, requester name, and keywords describing the metadata of all tasks in that group. As we vary the number of task groups and the number of tasks per group, we have #task groups $\times$ #tasks per task group $= |\mathcal{T}^i|$ tasks as input to each experiment.

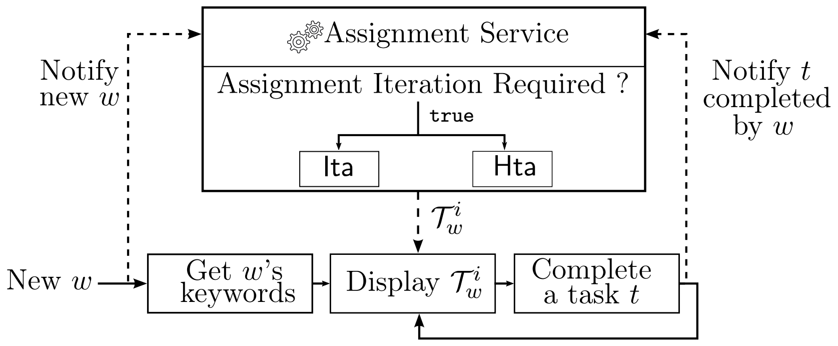


Figure A.1: GACS crowdsourcing platorm - Workflow

## A.2 Crowdsourcing Platform

To achieve adaptability, we develop a new platform, GACS (Grenoble Adaptive Crowd Sourcing), where workers are hired from an external source, AMT in our case. Figure A.1 illustrates a work session within GACS for a given worker $w$. First, we ask a worker to choose a minimum number of keywords to build her keyword vector. Then, we display the first set of tasks $\mathcal{T}_w^i$ assigned to $w$. At each iteration, $w$ is shown a new set of tasks using one the strategies that we investigate in the variants of motivation-aware task assignment Ita or Hta.

We design an assignment service that assigns task to a worker only if necessary. When a new worker arrives, we notify the assignment service that assigns a new set of tasks $\mathcal{T}_w^i$ to $w$. Each time $w$ completes a task, the assignment service is notified. This service monitors (i) the number of completed tasks for each worker and (ii) the overall number of completed tasks. It decides if a new assignment iteration must occur.

This decision depends on the problem that is considered. For Ita, where tasks are assigned to workers *individually*, a new assignment iteration is performed if the worker has completed at least $m$ tasks (e.g. 5). For Hta, an additional condition is used: an iteration occurs when at least $M$ tasks are completed *overall* since the previous iteration. Therefore, an iteration may occur every time $M$ are completed, and concerns only workers who completed at least $m$ tasks. Our rationale is: (i) to have a stable system, that does not re-assign tasks to workers too frequently and (ii) to get sufficient input to accurately estimate $\alpha_w^i$ or $\beta_w^i$. The additional condition in Hta allows to increase the probability to assign tasks to several workers *simultaneously* (i.e. $|\mathcal{W}^i| > 1$), which enables *holistic* task assignment.

The assignment service also manages the set of tasks available for assignment. For instance, when a worker is assigned a task $t$ at iteration $i$ but she does not complete it during this iteration, the service adds $t$ to the set of available tasks $\mathcal{T}_w^{i+1}$ in the next iteration.

**Worker Recruitment** To recruit workers on our platform, we publish *HITs* on Amazon Mechanical Turk (AMT) to recruit workers. Each HIT corresponds to a work session on our platform. When a worker accepts a HIT, she is asked to visit our web application. On our platform, she completes multiple tasks. When she terminates her work session, she get a verification code. Then, she pastes the code on AMT and submit the HIT for payment. Each HIT may be submitted by at most one worker. We pay a worker using (i) the HIT reward, that is given if a worker completed at least 1 task (on AMT, we "accept" her HIT) and (ii) a HIT *bonus*, that is equivalent to the total reward of the tasks she completed on our platform.

# Appendix B

# Proofs for Hta

## B.1 Proof of Equation 3.17

We aim to show:

$$\sum_{w \in \mathcal{W}^i} motiv(\mathcal{T}^i_w, w) = \sum_{\substack{k,l \in 1,\ldots,|\mathcal{T}^i| \\ k \neq l}} a_{\pi(k),\pi(l)} b_{k,l} + \sum_{k \in 1,\ldots,|\mathcal{T}^i|} c_{k,\pi(k)}$$

First, we have:

$$\begin{aligned}
&\sum_{\substack{k,l \in 1,\ldots,|\mathcal{T}^i| \\ k \neq l}} a_{\pi(k),\pi(l)} b_{k,l} \\
&= \sum_{\substack{\pi(k),\pi(l) \in \\ [\![1, X_{max}]\!]}} a_{\pi(k),\pi(l)} b_{k,l} + \sum_{\substack{\pi(k),\pi(l) \in \\ [\![X_{max}+1, 2X_{max}]\!]}} a_{\pi(k),\pi(l)} b_{k,l} \\
&+ \ldots + \sum_{\substack{\pi(k),\pi(l) \in \\ [\![(|\mathcal{W}^i|-1)X_{max}, |\mathcal{W}^i| X_{max}]\!]}} a_{\pi(k),\pi(l)} b_{k,l} \qquad \text{(B.1)} \\
&= \sum_{\substack{\pi(k),\pi(l) \in \\ [\![1, X_{max}]\!]}} \alpha^i_{w_1} d(t_k, t_l) + \sum_{\substack{\pi(k),\pi(l) \in \\ [\![X_{max}+1, 2X_{max}]\!]}} \alpha^i_{w_2} d(t_k, t_l) \\
&+ \ldots + \sum_{\substack{\pi(k),\pi(l) \in \\ [\![(|\mathcal{W}^i|-1)X_{max}, |\mathcal{W}^i| X_{max}]\!]}} \alpha^i_{w_{|\mathcal{W}^i|}} d(t_k, t_l) \\
&= 2 \sum_{\substack{t_k, t_l \in \mathcal{T}^i_{w_1} \\ k > l}} \alpha^i_{w_1} d(t_k, t_l) + 2 \sum_{\substack{t_k, t_l \in \mathcal{T}^i_{w_2} \\ k > l}} \alpha^i_{w_2} d(t_k, t_l) \\
&+ \ldots + 2 \sum_{\substack{t_k, t_l \in \mathcal{T}^i_{w_{|\mathcal{W}^i|}} \\ k > l}} \alpha^i_{w_{|\mathcal{W}^i|}} d(t_k, t_l) \qquad \text{(B.2)}
\end{aligned}$$

On line B.1, we decompose the sum on each sub-matrix in $A$. Cases that are ignored return 0. On line B.2, we use the definition of each $\mathcal{T}^i_w$ from equation 3.16. Additionally,

$$\begin{aligned}
\sum_{k \in 1,\dots,|\mathcal{T}^i|} c_{i,\pi(k)} &= \sum_{\substack{\pi(k)\in \\ [\![1,X_{max}]\!]}} c_{k,\pi(k)} + \sum_{\substack{\pi(k)\in \\ [\![X_{max}+1,2X_{max}]\!]}} c_{k,\pi(k)} \\
&+ \dots + \sum_{\substack{\pi(k)\in \\ [\![(|\mathcal{W}^i|-1)X_{max},|\mathcal{W}^i|X_{max}]\!]}} c_{k,\pi(k)} \qquad (B.3)\\
&= \sum_{t_k \in \mathcal{T}^i_{w_1}} \beta^i_{w_1} rel(w_1,t_k)(X_{max}-1) \\
&+ \sum_{t_k \in \mathcal{T}^i_{w_2}} \beta^i_{w_2} rel(w_2,t_k)(X_{max}-1) \\
&+ \dots + \sum_{t_k \in \mathcal{T}^i_{w_{|\mathcal{W}^i|}}} \beta^i_{w_{|\mathcal{W}^i|}} rel(w_{|\mathcal{W}^i|},t_k)(X_{max}-1) \qquad (B.4)
\end{aligned}$$

Summing B.2 and B.4, we obtain Equation 3.17.

## B.2 Proof of Theorem 5

We adapt the proof of Arkin et al. [10]. First, we show that the value of the optimal solution for Hta is less than 4 times the solution value for the auxiliary problem Lsap.

Let $M_B$ be a maximum matching in $B$. Let $\overline{M_B} = \{\{t_k,t_l\} \notin M_B,\ \exists t_{k'} s.t. \{t_k,t_{k'}\} \in M_B, \exists t_{l'} s.t. \{t_l,t_{l'}\} \in M_B\}$. Since $M_B$ is a maximum matching, we have

$$\forall t_k, t_l \in \overline{M_B}, d(t_k,t_l) \leq d(t_k,t_{k'}) + d(t_l,t_{l'}) \qquad (B.5)$$

Let $t_m$ be the task that is not incident to any edge in $M_B$ (there is at most such one, when $|\mathcal{T}^i|$ is odd). Let $E(t_m)$ the set of edges incident to $t_m$. Since $M_B$ is a maximum matching, we necessarily have:

$$\forall \{t_n,t_{n'}\} \in M_B, d(t_m,t_n) \leq d(t_n,t_{n'}) \wedge d(t_m,t_{n'}) \leq d(t_n,t_{n'}) \qquad (B.6)$$

Both Equations B.5 and B.6 also hold if $M_B$ is a *greedy* matching [10]. Let $\mathcal{Y} = \{1,\dots,|\mathcal{T}^i|\}$ and $\pi^*$ be the permutation of $\mathcal{Y}$ associated to the optimal solution hta* of Hta. We have:

$$\text{hta*} = \sum_{\substack{k,l\in\mathcal{Y} \\ k\neq l}} a_{\pi^*(k),\pi^*(l)} b_{k,l} + \sum_{k\in\mathcal{Y}} c_{k,\pi^*(k)} \qquad (B.7)$$

We decompose the first member of the sum:

$$\begin{aligned}
&\sum_{\substack{k,l\in\mathcal{Y}\\k\neq l}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l)\\
&= \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in M_B}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l) + \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in \overline{M_B}}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l)\\
&+ \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in E(t_m)}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l) \qquad\qquad\text{(B.8)}
\end{aligned}$$

Now,

$$\begin{aligned}
&\sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in \overline{M_B}}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l)\\
&\leq \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in \overline{M_B}}} a_{\pi^*(k),\pi^*(l)} (d(t_k,t_{k'}) + d(t_l,t_{l'})) \qquad\qquad\text{(B.9)}
\end{aligned}$$

where $\{t_k, t'_k\} \in M_B$ and $\{t_l, t'_l\} \in M_B$ (using Equation B.5). Additionally,

$$\sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in E(t_m)}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l) \leq \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in E(t_m)}} a_{\pi^*(k),\pi^*(l)} d(t_n,t'_n) \qquad\qquad\text{(B.10)}$$

where $\{t_n, t'_n\}$ is the edge in $M_B$ incident to $\{t_k, t_l\} \in \overline{M_B}$ (using Equation B.6). Combining Equations B.9 and B.10:

$$\begin{aligned}
&\sum_{\substack{k,l\in\mathcal{Y}\\k\neq l}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l)\\
&\leq 2\sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in M_B}} a_{\pi^*(k),\pi^*(l)} d(t_k,t_l) + \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in \overline{M_B}}} a_{\pi^*(k),\pi^*(l)} (d(t_k,t_{k'}) + d(t_l,t_{l'}))\\
&+ \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in E(t_m)}} a_{\pi^*(k),\pi^*(l)} d(t_n,t'_n)\\
&= \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in M_B}} d(t_k,t_l)(deg^A_{\pi^*(k)} + deg^A_{\pi^*(l)}) \qquad\qquad\text{(B.11)}
\end{aligned}$$

Using Equation B.11 in Equation B.7:

$$\begin{aligned}
\mathsf{hta}^* &\leq \sum_{\substack{k,l\in\mathcal{Y},\\\{t_k,t_l\}\in M_B}} d(t_k,t_l)(deg^A_{\pi^*(k)} + deg^A_{\pi^*(l)}) + 2\sum_{k\in\mathcal{Y}} c_{k,\pi^*(k)}\\
&= 2\sum_{k\in\mathcal{Y}} f_{k,\pi^*(k)} \qquad\qquad\text{(B.12)}
\end{aligned}$$

Let $\pi'^*$ the optimal solution of the Lsap instance (our auxiliary problem) and $\pi'$ the solution of GreedyMatching on this instance. Let lsap-sol the value of the solution $\pi'$ for Lsap. We have:

$$\text{lsap-sol} = \sum_{k \in \mathcal{Y}} f_{k,\pi'(k)} \geq \frac{1}{2} \sum_{k \in \mathcal{Y}} f_{k,\pi'^*(k)} \tag{B.13}$$

since GreedyMatching is a $\frac{1}{2}$-approximation for this Lsap instance. Therefore, we have

$$\begin{aligned} \text{hta*} &\leq 2 \sum_{k \in \mathcal{Y}} f_{k,\pi^*(k)} \\ &\leq 4 \sum_{k \in \mathcal{Y}} f_{k,\pi'(k)} = 4 * \text{lsap-sol} \end{aligned} \tag{B.14}$$

We now prove the second part of the proof. Let hta-gre-sol be the value of the solution returned by Hta-Gre for Hta. Let $\pi$ the permutation associated to this solution. We have:

$$\begin{aligned} \text{hta-gre-sol} &= \sum_{\substack{k,l \in \mathcal{Y} \\ k \neq l}} a_{\pi(k),\pi(l)} b_{k,l} + \sum_{k \in \mathcal{Y}} c_{k,\pi(k)} \\ &= \sum_{\substack{k,l \in \mathcal{Y}, \\ \{t_k,t_l\} \in M_B}} a_{\pi(k),\pi(l)} d(t_k,t_l) + \sum_{\substack{k,l \in \mathcal{Y}, \\ \{t_k,t_l\} \in \overline{M_B}}} a_{\pi(k),\pi(l)} d(t_k,t_l) \\ &+ \sum_{\substack{k,l \in \mathcal{Y}, \\ \{t_k,t_l\} \in E(t_m)}} a_{\pi(k),\pi(l)} d(t_k,t_l) + \sum_{k \in \mathcal{Y}} c_{k,\pi(k)} \end{aligned} \tag{B.15}$$

In Lines 12-16, Hta-Gre sets $(\pi(k),\pi(l))$ to $(\pi'(k),\pi'(l))$ or to $(\pi'(l),\pi'(k))$ for each $\{t_k,t_l\} \in M_B$. Each case occurs with probability $\frac{1}{2}$. In Equation B.15, $a_{\pi(k),\pi(l)} = a_{\pi'(k),\pi'(l)}$ ($A$ is symmetric and $t_k, t_l \in M_B$). Therefore, the expected contribution of pair $(k,l) \in \mathcal{Y}^2, \{t_k,t_l\} \in M_B$ is $a_{\pi'(k),\pi'(l)} * d(t_k,t_l)$. In Equation B.15, each $d(t_k,t_l)$ is multiplied by $a_{\pi'(k),\pi'(l)}$ or $a_{\pi'(k'),\pi'(l)}$ or $a_{\pi'(k),\pi'(l')}$ or $a_{\pi'(k'),\pi'(l')}$ where $\{t_k,t_l\} \in \overline{M_B}, \{t_k,t_{k'}\} \in M_B, \{t_l,t_{l'}\} \in M_B$. Equivalently, $a_{\pi'(k),\pi'(l)}$ is multiplied by $d(t_k,t_l)$ or $d(t_{k'},t_l)$ or $d(t_k,t_{l'})$ or $d(t_{k'},t_{l'})$, each case with probability $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Thus, the expected contribution of pair $(k,l) \in \mathcal{Y}^2, \{t_k,t_l\} \in \overline{M_B}$:

$$\begin{aligned} &\frac{1}{4} a_{\pi'(k),\pi'(l)} (d(t_k,t_l) + d(t_{k'},t_l) + d(t_k,t_{l'}) + d(t_{k'},t_{l'})) \\ &\geq \frac{1}{4} a_{\pi'(k),\pi'(l)} \times 2 \max\{d(t_k,t_{k'}), d(t_l,t_{l'})\} \\ &\geq \frac{1}{4} a_{\pi'(k),\pi'(l)} (d(t_k,t_{k'}) + d(t_l,t_{l'})) \end{aligned} \tag{B.16}$$

In Equation B.15, each $a_{\pi'(k),\pi'(l)}$ where $k = m$ or $l = m$ (suppose $k = m$) is multiplied by $d(t_m,t_l)$ or $d(t_m,t'_l)$ where $\{t_l,t_{l'}\} \in M_B$, each with probability $\frac{1}{2}$. Therefore, the expected contribution of pair $(k,l) \in \mathcal{Y}^2, \{t_k, t_l\} \in E(t_m)$ is:

$$\frac{1}{2}a_{\pi'(k),\pi'(l)}(d(t_m,t_l)+d(t_m,t'_l)) \geq \frac{1}{2}a_{\pi'(k),\pi'(l)} * d(t_l,t_{l'}) \tag{B.17}$$

since $d()$ satisfies the triangle inequality. In B.15, the expected contribution of $c_{k,\pi(k)}$ is at least $\frac{1}{2}c_{k,\pi'(k)}$ since $\pi(k)=\pi'(k)$ with probability $\frac{1}{2}$. If we combine the previous propositions in Equation B.15, we obtain:

$$\begin{aligned}\textsf{hta-gre-sol} &\geq \frac{1}{4}\sum_{\substack{k,l\in\mathcal{Y},\\ \{t_k,t_l\}\in M_B}} d(t_k,t_l)(deg^A_{\pi'(k)}+deg^A_{\pi'(l)})+\frac{1}{2}\sum_{k\in\mathcal{Y}}c_{k,\pi'(k)}\\ &=\frac{1}{2}\sum_{k\in\mathcal{Y}}f_{k,\pi'(k)} \qquad (B.18)\end{aligned}$$

Combining Equations B.14 and B.18, we prove:

$$\textsf{hta-gre-sol} \geq \frac{1}{2}\textsf{lsap-sol} \geq \frac{1}{8}\textsf{hta*}$$