Search and Aggregation in Big Graphs

Abdelmalek Habi

N°d'ordre NNT : 2019LYSE1259

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON

Opérée au sein de :

l'Université Claude Bernard Lyon 1

Ecole Doctorale N° 512

Informatique et Mathématiques (InfoMaths)

Spécialité de doctorat : Informatique

Soutenue publiquement le 26/11/2019, par :

Abdelmalek HABI

Search and Aggregation in Big Graphs

Devant le jury composé de :

To my mother & father.

To my wife & daughter.

To my sisters & brothers.

Acknowledgements

It would not have been possible to achieve this thesis and write this manuscript without the help and support of the kind people around me, to only some of whom it is possible to give particular mentions here.

First, I wish to express my sincere gratitude to my supervisors M. Hamamache Kheddouci and M. Brice Effantin, for their excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research. I thank them for their continuous support and encouragement on both my research and other matters of life during my Ph.D. study. As advisors, they taught me practices and skills that will benefit my future academic career. It has been a great fortune for me to work under their supervision.

Many thanks go to Mme. Zoubida Kedad and M. Olivier Togni for the time they spent for reviewing my thesis manuscript, and for their valuable comments and constructive suggestions on the thesis.

I am grateful to the members of the defense committee: Mme. Angela Bonifati, M. Mohand Boughanem, Mme. Zoubida Kedad and M. Olivier Togni for their friendship and wisdom.

I express my love and gratitude to my parents, my wife, my sisters, my brothers and all my family for their continuous moral support and encouragement with their best wishes. Their love accompanies me wherever I go.

Finally, I especially thank all my lab friends, whose wonderful presence made it a convivial place to work. I warmly thank all my friends who were always supporting and encouraging me with their best wishes.

Abstract

Recent years have witnessed a growing renewed interest in the use of graphs as a reliable means for representing and modeling data. Thereby, graphs enable to ensure efficiency in various fields of computer science, especially for massive data where graphs arise as a promising alternative to relational databases for big data modeling. In this regard, querying data graph proves to be a crucial task to explore the knowledge in these datasets.

In this dissertation, we investigate two main problems. In the first part we address the problem of detecting patterns in larger graphs, called the top-k graph pattern matching problem. We introduce a new graph pattern matching model named *Relaxed Graph Simulation (RGS)*, to identify significant matches and to avoid the empty-set answer problem. We formalize and study the top-k matching problem based on two classes of functions, relevance and diversity, for ranking the matches according to the *RGS* model. We also consider the diversified top-k matching problem, and we propose a diversification function to balance relevance and diversity. Moreover, we provide efficient algorithms based on optimization strategies to compute the top-k and the diversified top-k matches according to the proposed model. The proposed approach is optimal in terms of search time and flexible in terms of applicability. The analyze of the time complexity of the proposed algorithms and the extensive experiments on real-life datasets demonstrate both the effectiveness and the efficiency of these approaches.

In the second part, we tackle the problem of graph querying using aggregated search paradigm. We consider this problem for particular types of graphs that are trees, and we deal with the query processing in XML documents. Firstly, we give the motivation behind the use of such a paradigm, and we explain the potential benefits compared to traditional querying approaches. Furthermore, we propose a new method for aggregated tree search, based on approximate tree matching algorithm on several tree fragments, that aims to build, the extent possible, a coherent and complete answer by combining several results. The proposed solutions are shown to be efficient in terms of relevance and quality on different real-life datasets.

Keywords: graph matching, graph pattern matching, graph simulation, relaxed graph simulation, top-k, diversified top-k, aggregated search, tree matching

Résumé

Ces dernières années ont connu un regain d'intérêt pour l'utilisation des graphes comme moyen fiable de représentation et de modélisation des données, et ce, dans divers domaines de l'informatique. En particulier, pour les grandes masses de données, les graphes apparaissent comme une alternative prometteuse aux bases de données relationnelles. Plus particulièrement, le recherche de sous-graphes s'avère être une tâche cruciale pour explorer ces grands jeux de données.

Dans cette thèse, nous étudions deux problématiques principales. Dans un premier temps, nous abordons le problème de la détection de motifs dans les grands graphes. Ce problème vise à rechercher les k-meilleures correspondances (top-k) d'un graphe motif dans un graphe de données. Pour cette problématique, nous introduisons un nouveau modèle de détection de motifs de graphe nommé la *Simulation Relaxée de Graphe* (*RGS*), qui permet d'identifier des correspondances de graphes avec un certain écart et ainsi éviter le problème de réponse vide. Ensuite, nous formalisons et étudions le problème de la recherche des k-meilleures réponses suivant deux critères, la pertinence (la meilleure similarité entre le motif et les réponses) et la diversité (la dissimilarité entre les réponses). Nous considérons également le problème des k-meilleures correspondances diversifiées et nous proposons une fonction de diversification pour équilibrer la pertinence et la diversité. En outre, nous développons des algorithmes efficaces basés sur des stratégies d'optimisation en respectant le modèle proposé. Notre approche est efficiente en terme de temps d'exécution et flexible en terme d'applicabilité. L'analyse de la complexité des algorithmes et les expérimentations menées sur des jeux de données réelles montrent l'efficacité des approches proposées.

Dans un second temps, nous abordons le problème de recherche agrégative dans des documents XML. Pour un arbre requête, l'objectif est de trouver des motifs correspondants dans un ou plusieurs documents XML et de les agréger dans un seul agrégat. Dans un premier temps nous présentons la motivation derrière ce paradigme de recherche agrégative et nous expliquons les gains potentiels par rapport aux méthodes classiques de requêtage. Ensuite nous proposons une nouvelle approche qui a pour but de construire, dans la mesure du possible, une réponse cohérente et plus complète en agrégeant plusieurs résultats provenant de plusieurs sources de données. Les expérimentations réalisées sur plusieurs ensembles de données réelles montrent l'efficacité de cette approche en termes de pertinence et de qualité de résultat.

Mots clés: appariement de graphes, recherche de motifs de graphe, simulation de graphes, simulation relaxée de graphes, top-k, top-k diversifiés, recherche agrégative dans les graphes,

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Graphs are a useful and powerful paradigm for formalizing problems and modeling complex and heterogeneous data with their relationships. They consist of a set of vertices, which represent objects, and a set of edges, which represent relations between these objects. They have attracted the interest of the whole scientific community, allowing their use in a wide range of problems in intelligent information processing such as pattern recognition [1], information retrieval [2], knowledge discovery [3], dynamic network traffic [4], social network analysis [5], protein interactions [6] and so on.

With the advent of recent applications, data graphs are growing in size exponentially, with millions of nodes and billions of edges. Graph search is a primordial task to identify the occurrences of the information sought. In fact, graph search is a challenging issue. It describes the act of searching for given information, usually in the form of a query graph, in a given graph, often called the target graph. This process can be performed by searching for subgraphs using graph matching and more generally graph comparison. Typically, this problem is defined as the process of finding a mapping between components of two graphs that provides the best alignment of their (sub)structures. Graph matching solutions are classified into two broad categories: exact approaches and inexact approaches. Exact matching approaches return (sub)graphs that match exactly the given query. Inexact matching approaches look for results even if they are structurally different, to some extent, from the query.

1.1 Thesis scope

In this thesis, inexact graph matching and its applications for querying data graphs are investigated. Thus, the thesis is divided into two main parts: the graph matching problem in large graphs and the distributed tree querying problem, also known as aggregated tree search.

Part I: The graph matching problem

Graphs are powerful mathematical structures constituting universal modeling and representation tools used in a wide range of real-world applications. Actually, the success of such graph-based applications depends on the performance of the underlying graph query processing. In this context, finding graph matching solutions that guarantee optimality in terms of accuracy and time complexity is a challenging issue. Thus, various types of graph matching have been widely investigated. Graph pattern matching (GPM) is among the most important challenges of graph processing and plays a central role in various emerging applications. Usually, GPM is defined in terms of subgraph isomorphism [7], which seeks subgraphs that are exactly isomorphic to the query graph, or graph simulation [8], which has been adopted for graph pattern matching to cope with restrictions of exact graph matching approaches.

However, the actual sheer increase in data introduces new challenges to graph pattern matching, from its definition to corresponding processing methods. The main problem in nowadays real-life graphs is the volume of information to be processed, and the vast search space in which the search process is performed. For instance, Facebook amounts to 1.562 million daily active users [9] and Twitter totals up to 330 million active users [10]. It is prohibitively expensive to query such large graphs. The subgraph isomorphism is an NP-*complete* problem [11] and the main issues are that the search process is too costly, too restrictive and does not scale well [12]. Whereas graph simulation has a quadratic time, this notion is often too restrictive to match queries in newly emerging fields such as social network analysis [13].

Dealing with such a massive size of data gives rise to other problems such as the excessive number of returned results. Using the subgraph isomorphism, the set of matches may contain exponentially many subgraphs of the target graph [14]. Besides, the size of the result set, using graph simulation, depends on the size of the target graph and the query [8]. Inspecting all the results to find what users are looking for is a daunting task. Thus, users are generally interested in responses that satisfy their preferences, which are usually relevance and diversity [15].

In the first part of this work, we define and formalize the graph pattern matching problem using a new notion of graph simulation, called *Relaxed Graph Simulation (RGS)*, based on query relaxation, which allows reaching more significant matches and coping with the empty-set answer problem. We also investigate the top-k matching problem based on two function classes, relevance and diversity, for ranking the matches with respect to the proposed model. Furthermore, we study the diversified top-k matching problem, and we propose a diversification function to balance relevance and diversity. Nonetheless, we provide efficient algorithms based on optimization strategies to compute the top-k and diversified top-k matches according to the *RGS* model.

Part II: Aggregated tree search

Information retrieval can be defined as the process that links some information material, from a large collection of information resources, to an information need expressed by the user [16]. Nowadays, XML (eXtensible Markup Language) standard [17] is one of the most used formats for representing and exchanging information. The simple nature, the self-description and the ability to describe a wide range of data brought XML to greater importance. The sheer increase in the generation and use of XML documents leads to the need for appropriate retrieval approaches that can exploit the specific features of this kind of documents. Numerous approaches have been proposed to deal with XML retrieval. They usually use the tree representation of documents and queries to process them [18]. Indeed, the retrieval process can be considered as a tree matching problem between the query tree and the document trees.

Currently, information retrieval systems have evolved from the document-level access to the in-depth search methods, *i.e.,* seeking specific required document components (*i.e.,* documents parts) instead of entire documents. Their purpose is then to return a ranked list of answers that are deemed relevant to the query. However, returning and presenting results in this way are often not appropriate to the user's expectations [19–21]; especially when the expected needs are scattered across several data sources and/or several documents. Thus, there is a need for more focus, more organization, and more diversity in the returned results. Besides, approaches that process information with finer granularity and build a response by combining multiple contents that may be useful to the user are needed. Aggregated search addresses these tasks of searching and assembling results.

Several paradigms of graph querying have been proposed in the literature, but only a few of them have tackled the problem of aggregated search. In the second part of this thesis, we introduce and develop approaches and techniques to deal with this problem. We consider this problem for trees, which are a particular class of graph, and we propose a framework for XML querying based on aggregated tree search. This problem becomes more challenging when considering the complex representation of data in graph form. However, it is more difficult for distributed graphs.

1.2 Thesis organization

The remaining of this thesis contains six chapters: The first part of this thesis, Chapters 2 and 3, is devoted to the graph matching problem. While the second part of the thesis, from Chapter 4 through 6, is about the aggregated tree search.

In Chapter 2 *'Graph search methods'*, we first give some preliminaries and we introduce some notations needed in the rest of this thesis. Then, we highlight the most important research directions related to the graph querying problem.

In Chapter 3 *'Diversified top-k search with relaxed graph simulation'*, we present our first contribution in which we propose a new notion of graph pattern matching and we study (diversified) top-k graph pattern matching problem.

In Chapter 4 *'Aggregated search: Definition and overview'*, we give an overview of the general aggregated search concept and its related work.

In Chapter 5 *'Tree matching and XML retrieval'*, we outline and compare the most important approaches for XML retrieval using tree matching.

In Chapter 6 *'A new approximate XML retrieval based on aggregation search'*, we present our proposed framework for performing aggregated tree search in XML documents.

Finally, in Chapter 7, we conclude the manuscript by summarizing the major contributions of this thesis and raising important future work directions and perspectives.

Chapter 2

Graph search methods

Contents

Graphs are efficient data structures that model items with their relationships. An item, called node or vertex, represents an object (data), and a link between two items, called edge, models a relationship between two objects (data). Graphs represent an effective way of formalizing problems and representing objects used to represent complex and heterogeneous data in numerous domains. The flexible nature of graphs allows adding new relationships and even new objects without affecting existing application functionalities and queries. This flexibility brought graphs to greater importance, especially for massive data. They are more and more used with the advent of modern applications ranging from scientific databases or biological networks to connected world with social networks. In many of these applications, graphs are huge, with millions of nodes and billions of edges, and it is difficult to mine their data. In this context, the graph querying (or the graph search) is usually used to identify the occurrences of a query. Thus various types of matchings have been widely investigated. In this chapter, we discuss graph querying and survey some efficient and recent graph search methods. We first introduce some useful definitions related to graphs and we present some common concepts and typical classes of graphs. We also give definitions of some theoretical terms and concepts used in the context of graph search. Then, we highlight the most important research directions related to the graph querying problem.

2.1 Basic definitions

In this section, we introduce and define some key concepts and notations relating to graphs.

Graph. A graph G is a four-tuple $G = (V(G), E(G), l, \Sigma)$, where (1) $V(G)$ is a finite not empty set of nodes (also called vertices), (2) $E(G) \subseteq V(G) \times V(G)$ is the set of edges in which (u, v) denotes an edge from u to v, (3) $l : V(G) \cup E(G) \longrightarrow \Sigma$ is a labeling function on the nodes and edges, such that for each node (edge) $v \in V(G)$ (resp. $e \in E(G)$), $l(v)$ (resp. $l(e)$) is a label from the finite set of labels Σ. The cardinality of the node set $V(G)$ is called the *order* of G, commonly denoted by $|V(G)|$, and the cardinality of the edge set $E(G)$, denoted $|E(G)|$, is the *size* of G.

Graphs are used to formalize and to represent complex data where nodes and edges represent respectively items and relations between these items. A graph is said *directed* if the edges have a direction associated with the nodes, *i.e.,* edges are ordered pairs (u, v) connecting the source node u to the target node v, otherwise the graph is *undirected*, *i.e.,* edges are unordered pairs $\{u, v\}$ and connect the two nodes in both directions. In an undirected graph G, two distinct nodes u and v are adjacent (or *neighbors*) if there exists an edge $(u, v) \in E(G)$ that connects them. An edge (u, v) is said to be incident to the nodes u and v. A graph is said *labeled* if for its components there is a labeling assignment, *i.e.,* nodes and/or edges are labeled by assigning one or more values (symbolic or numeric); otherwise the graph is *non-labeled*. However, non-labeled graphs can be considered as a special case of labeled graphs where the labels of all nodes and edges are identical or null. Figure 2.1 shows some graph examples.

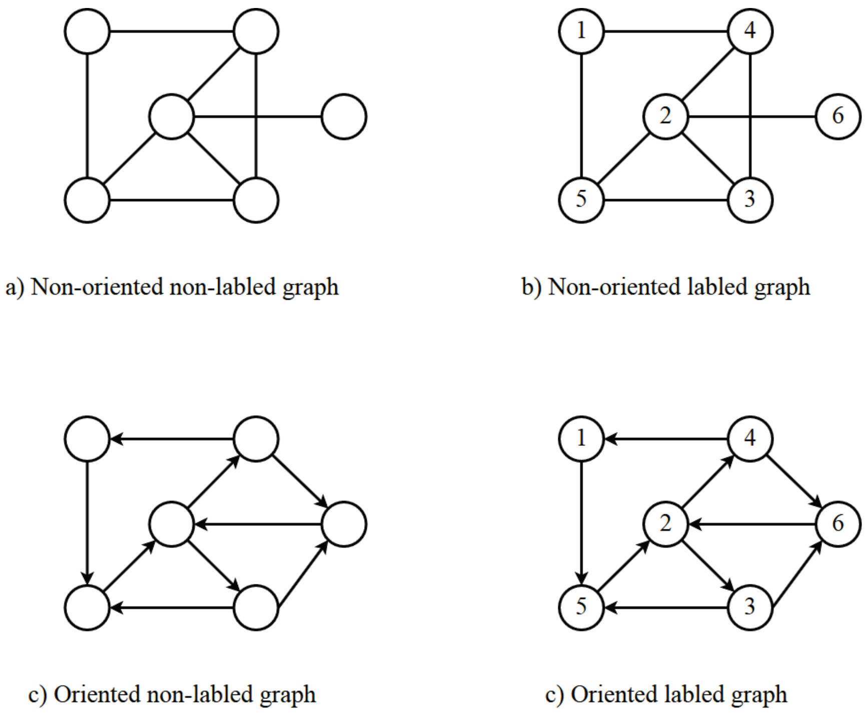

1
4
2
6
5
3
a) Non-oriented non-labled graph
b) Non-oriented labled graph
1
4
2
6
5
3
c) Oriented non-labled graph
c) Oriented labled graph

FIGURE 2.1: Example of graphs.

Degree and Neighborhood. The set of all neighbors of a node $v \in G$ is denoted $N(v)$. The *degree* of a node u, denoted $deg(u)$, is the number of its neighbors. If $deg(v) = 0$ then the node v is an *isolated node*, *i.e.*, v is not adjacent to any other vertex. A node of degree one ($deg(v) = 1$) is called an endpoint or pendant node. The minimum degree of a graph G is $\delta(G) = \min\{deg(v) : v \in V(G)\}$ and the maximum degree of a graph G is $\Delta(G) = \max\{deg(v) : v \in V(G)\}$

Subgraphs. A graph that is contained in another graph is called a *subgraph*. A graph $G' = (V(G'), E(G'), l', \Sigma)$ is a subgraph of $G = (V(G), E(G), l, \Sigma)$, denoted by $G' \subseteq G$, if $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$, $l'(x) = l(x) \forall x \in V(G)$, and $l'(e) = l(e) \forall e \in E(G)$. A graph G' is a *spanning subgraph* of G if G' contains all the nodes of G. For a subgraph G', if $E(G')$ contains all the edges of $E(G)$ that have endpoints in $V(G')$, then G' is an *induced subgraph* of G. Otherwise G' is a *partial subgraph*. Figure 2.2 illustrates the concepts of induced and partial subgraphs.

We refer the interested reader to [22–24] and references therein for more background information on graph theory.

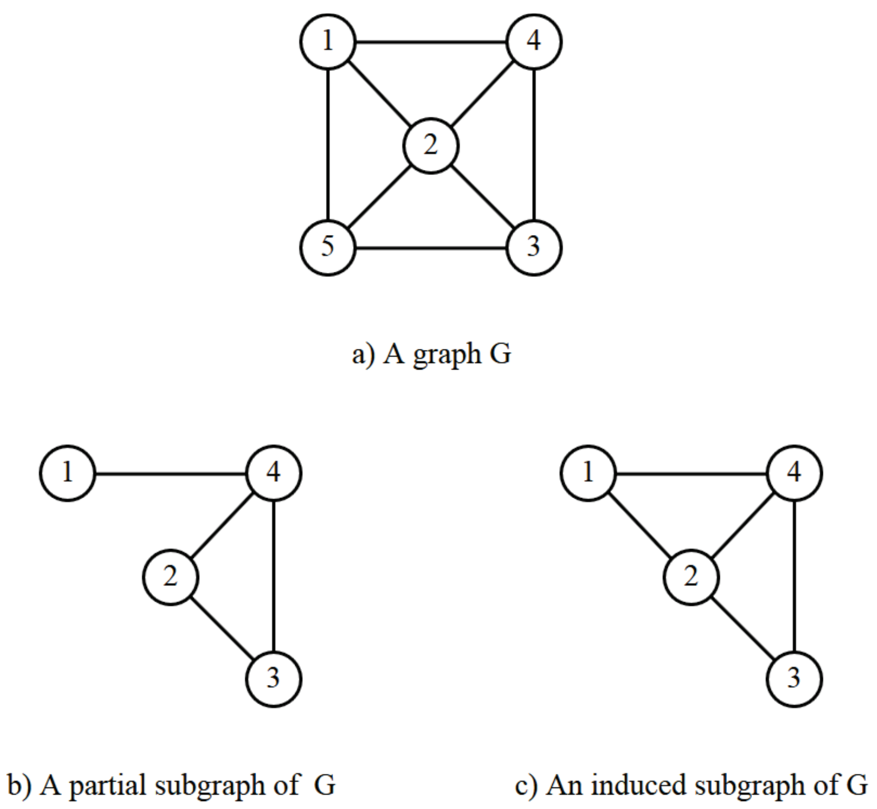


Figure 2.2: Example of induced and partial subgraphs.

2.2 Querying data graph

Graphs are a universal and flexible paradigm for representing and modeling data and their relationships. Thus various real applications such as protein interactions and social networks use graphs as a model of representation and searching. Usually, the success of such an application depends mainly on the efficiency and the quality of the underlying graph query processing. Graph querying (or graph search) is the task of searching for given information, usually in the form of a query graph, in a given graph, often called the target graph. Talking about graph querying problem leads directly to one of the most prevalent problems in the field of graph theory, which is graph matching. Typically, the problem of graph matching is defined as the process of finding a mapping between the nodes of two graphs that provides the best alignment of their structures.

In most real-world applications, data graphs are enormous, with millions of nodes and billions of edges, and it is difficult to mine their data. In this context, a query graph is usually used to identify the occurrences for the given information. Thus various types of matchings have been widely investigated. Graph pattern matching is being one of the fundamental tasks on which are based the search, the querying, and the analysis of data graph. Typically, graph pattern matching methods are classified into two broad categories according to their results. The first category represents exact matching approaches, which return graphs or subgraphs that match exactly the given query. The second category concerns inexact matching approaches, in which the returned results may be structurally different from the query to some extent. In Section 2.2.1 and Section 2.2.2, we present respectively exact and inexact graph pattern matching algorithms.Then, in Section 2.2.3, we present graph simulation, which has been adopted for graph pattern matching to cope with restrictions of existing paradigms.

2.2.1 Exact graph matching

Exact graph matching methods aim to find out an exact mapping between the nodes and the edges of the compared graphs or at least between subparts of them. In other words, with exact graph matching, edge-preserving must be ensured, *i.e.,* if two nodes in the first graph are linked by an edge, their correspondents in the second graph are also linked by an edge. Graph isomorphism represents the most stringent form of graph matching, in which the mapping is a bijective correspondence and the edge-preserving is satisfied in both directions.

Definition 1. Let $G = (V(G), E(G), l, \Sigma)$ and $G' = (V(G'), E(G'), l', \Sigma)$ be two graphs. G and G' are isomorphic if there exists a bijective function h: $V(G) \longrightarrow V(G')$ such that: (1) $\forall u \in V(G) : l(u) = l'(h(u))$, (2) $\forall (u, v) \in E(G) : (h(u), h(v)) \in E(G')$ and $l((u, v)) = l'((h(u), h(v)))$ and (3) $\forall (h(u), h(v)) \in E(G') : (u, v) \in E(G)$ and $l'((h(u), h(v))) = l((u, v))$.

Subgraph isomorphism [7] can be considered as a weaker form of the exact graph matching, in which an isomorphism holds between one of the two graphs and a subgraph of the other graph. Other forms of exact graph matching exist, in which subgraph isomorphism is used in a slightly weaker sense as in [25]. In other words, the constraint of edge-preserving in both directions is dropped. We give in the following a brief description of some of these approaches.

Graph monomorphism [26] is a relaxation of subgraph isomorphism where the mapped subgraph may have both extra nodes and extra edges. In other words, additional edges are allowed between nodes in the larger graph. Another variant is graph homomorphism, in which there is a mapping f from the node set of the first graph G to the node set of the other graph G' such that $(u, v) \in E(G)$ implies $(f(u), f(v)) \in E(G')$ but not vice versa. Finally, the maximum common subgraph (MCS) [27] is another interesting matching variant. MCS is defined as the problem of mapping a subgraph of one graph to an isomorphic subgraph of the other graph. Typically, the problem is to find the largest part of two graphs that is identical in term of structure.

It is important to highlight that all the forms of exact graph matching, that we cited before, belong to the NP-complete class. However, the graph isomorphism has not yet been demonstrated if it belongs or not to NP class [1]. For some special classes of graph, polynomial isomorphism algorithms have been developed (planar graphs [28], trees [29]). Recently, the author of [30] shows that graph isomorphism can be solved in quasi-polynomial time $(\exp((\log n)^{O(1)}))$. However, no polynomial algorithms are known for the general case.

Hence, the exact graph matching has exponential time complexity in the worst case. Consequently, using exact methods to deal with small graphs can be still acceptable. In the following, we briefly review some typical exact graph matching methods.

2.2.1.1 Exact subgraph matching methods

The problem of graph querying has been well studied and it has a rich history in various scenarios, among which protein interactions, social network analysis and graph database management play an important role. Typically, it can be handled by exact and approximate approaches. So in this section, we describe some graph querying approaches that are based on exact subgraph matching.

Tree search-based techniques have attracted much attention since the first proposal of Ullmann's algorithm [7] that addresses all forms of exact graph matching. These techniques represent the central pillar of the most existing algorithms for exact graph matching. Usually, they use a backtracking process in addition to some heuristics.

Using Ullmann's algorithm [7], the search process can be performed in two main steps: tree-search and refinement procedure. Firstly, for each query node, the algorithm seeks a set of candidate nodes. Then, it invokes a recursive subgraph search subroutine to find a potential mapping between a query node and a data node. All mapped data nodes are stored as the output of the first step, and they will be exploited in the refinement procedure. This latter aims to reduce the search space in order to minimize the computation time required for the subgraph isomorphism testing. To this end, the algorithm filters out candidate nodes that have a smaller degree than its correspondent node in the query graph. For each incident edge of a node kept as a candidate, the algorithm checks if there is a corresponding edge in the data graph.

A significant number of exact graph matching improvements have been proposed since the first proposition of Ullmann [7]. The *VF* algorithm [31] is designed to deal with both isomorphism and subgraph isomorphism. This algorithm describes a fast heuristic which analyses the nodes adjacent to the ones already added in the partial matching. After some years, the same authors propose a new algorithm called *VF2* [32], which is an improved version of the previous one. The *VF2* algorithm reduces the space complexity from $O(n^2)$ to $O(n)$, where n represents the number of nodes in the graphs. This algorithm defines a new concept s, called the state space representation, which represents a partial solutions of the correspondence between two graphs. The transition from the state s to its successor s' in the set of the partial solution M corresponds to a new pair of matching nodes. By the way, the algorithm starts with the first node, selects a connected node from the set of matched query nodes, looks for a subgraph match and backtrack if and when the need arises to do that. The main difference from the Ullman's algorithm is in the refinement step, where *VF2* algorithm requires two kinds of rules: syntactic feasibility rules and semantic feasibility rules.

More recent interesting algorithms have been proposed. The *Spath* algorithm [33] uses paths as patterns of comparison. It looks for matching paths instead of single nodes. It uses a path-signature function to minimize the search space. For each graph node and query node, *Spath* computes a neighborhood signature which makes it possible to decide if a given candidate node must be pruned or not. The authors introduce in [34] an algorithm, called *GADDI*, which firstly computes a neighborhood discriminating structure distance between pairs of adjacent nodes of the data graph. After that, the algorithm invokes a subgraph matching subroutine, which performs a two-way pruning, based on the above distance, and incorporates a dynamic matching schema. Ullmann in his new proposal [35] presents an important enhancement of his isomorphism algorithm [7]. The main idea of this new algorithm is based on the Binary Constraint Satisfaction Problem. *Turbo-iso* [36] is one of the most efficient algorithms that deal with the subgraph isomorphism problem. It focuses on solving the matching order selection problem. Proposers of *Turbo-iso* algorithm introduce two novel concepts, candidate region exploration and the combine and permute strategy. The candidate region exploration identifies candidate regions, which are subgraphs of the data graph where there is more chance to find embeddings for the query graph. Next, it computes a matching order for each candidate region explored. The combine and permute strategy exploits the novel concept of the neighborhood equivalence class (NEC). Each query node in the same NEC has identically matching data nodes. Finally, according to the obtained matching order and using the candidate data nodes of the NEC nodes, a recursive subroutine is processed for the subgraph search. In [37], the authors propose the *CFL-match* algorithm by postponing the Cartesian products based on the structure of a query to minimize the redundant Cartesian products. Based on the spanning tree of the query graph, the latter is decomposed into substructures by a Core-forest Decomposition method. Then, the subgraph matching is performed on each of these substructures. Briefly, the algorithm uses first the neighborhood label frequency filter to ensure that a data node is a deemed candidate. Then, it applies a second filter based on the maximum Neighbor-Degree to reduce the time processing of the first filter.

2.2.1.2 Other methods and techniques

Other methods addressing the problem of subgraph isomorphism search have been proposed in the literature. In the following, we give an overview of some of them. Nauty's algorithm [38] is one of the most efficient approaches, which is not based on tree search techniques. Nauty's algorithm constructs an automorphism group of each graph. Using transformation rules, it reduces graphs to canonical forms that may be checked relatively quickly for isomorphism. Two graphs are said isomorphic if their canonical forms are equal. The authors propose in [39] an isomorphism approach based on Random Walks. In [40], the authors propose algorithms to deal with graph isomorphism and subgraph isomorphism for graphs having unique node labels. *TMODS* [41] algorithm uses a set of genetic algorithms to find exact and inexact pattern matches in directed attributed graph. The study given in [42] presents a technique for speeding up the subgraph isomorphism on large graphs.

2.2.2 Inexact graph matching

The stringent conditions imposed by the exact graph matching paradigm and the high computational complexity make it too restrictive for graph querying. In many real-world applications, graphs are subject to deformations due to several causes, such as the noise in the acquisition process, the missing or incomplete information, or the errors introduced by the modeling processes. So, the obtained result graphs (subgraphs) are likely different from the query graph and its expected answers.

So, the matching process should be able to consider correspondences that do not satisfy, to some extent, all the requirements imposed by the query graph. Besides, the matching process must be able to reach a good approximate solution in an acceptable time, even without a guarantee to give the best solution. These reasons highlight the need for approximate (inexact) graph matching algorithms [43], also known as error-correcting matching [44] or error-tolerant matching [45]. Contrary to the exact graph matching approaches, the aim for inexact graph matching approaches is not to make a strict interpretation of structural constraints, but to select and rank subgraphs according to their probability to match the query graph. In other words, algorithms of this category aim to find a matching that minimizes the dissimilarity between the query graph and the subgraphs of the data graph.

Inexact graph matching algorithms can be classified into optimal and approximate algorithms. Optimal inexact matching algorithms can be considered as a generalization of the exact graph matching algorithms. They seek an exact solution achieving the global minimum of the matching cost if it exists. Generally, the algorithms of this class are not suitable for most applications due to their excessive processing costs. Approximate or suboptimal matching algorithms seek for a local minimum of the matching cost, which is not far from the global one. In such a class, there are no guarantees to reach the exact solution even if it exists. However, the main advantage of these algorithms is their shorter matching time, which is usually polynomial. In the following, we present and overview some important inexact graph matching approaches.

2.2.2.1 Graph edit distance based techniques

There are numerous approaches for inexact graph matching in the literature of querying data graphs. As we said above, the aim of these approaches is not to make a strict interpretation of structural constraints, *i.e.*, they do not impose the edge-preservation constraint used on exact matching. Typically, the inexact graph matching problem can be formulated as the problem of calculating the dissimilarity between two graphs. This problem can be reduced to the Graph Edit Distance (GED) problem [46, 47], where the matching cost is defined as a set of graph edit operations (node/edge insertion, node/edge deletion, etc.) between two graphs. Each edit operation has a predefined cost that contributes to the calculation of the edit distance between the two compared graphs. The edit distance is defined by the minimum cost sequence of edit operations that transform a graph into another [48].

Seeking the exact value of editing distance is not easy and can usually be expensive. This problem is NP-Complete [48] for general graphs and induces exponential computation time complexity [49]. Recently, several approximate methods have been proposed to deal with this issue [50]. They approach the exact value of GED in polynomial time, using different techniques, such as dynamic programming and bipartite assignment. In the following, we give an overview of two approximate formulations: bipartite assignment [51, 52] and quadratic assignment formulation [53, 54].

Bipartite based GED approaches have shown their efficiencies to solve error-tolerant matching [49, 51, 55, 56]. Mainly, they partition the compared graphs into smaller substructures and approximate the GED by a linear assignment problem. The latter can be solved efficiently via for instance the Hungarian algorithm [57] or the Jonker-Volgenant algorithm [58].

The quadratic-assignment-formulation [59] based GED methods show a significant improvement over the bipartite based GED methods. Using the definition of fuzzy paths, the authors show in [53] that the formulation based on the quadratic assignment programming problem (QAP) is suitable for the approximation of GED. In [54], the authors argue that this problem is mainly related to weighted graph matching problem [60], and they give a further formal and more general analysis on the transformation rules of GED to QAP form.

Further details about the problem could be found in [50] as well as references therein.

2.2.2.2 Tree search-based techniques

Inspired by the exact graph matching approaches, some works in the literature propose tree search-based techniques with backtracking to deal with the inexact graph matching. The search process is mainly based on heuristics that use the cost of the current partial matching and the estimated cost of the rest of the nodes. Thus, the resulting cost is used to prune unfruitful paths or to specify the traversal order in the search tree, as in the *A** algorithm [61]. Several works based on tree search have been proposed, for example, but not limited to [26, 62, 63]. In this research area, the *A** algorithm has shown its effectiveness, as presented in some works such as [64–66].

2.2.2.3 Other techniques

Given the importance of the graph matching problem and its involvement in a wide range of real-world applications, a significant number of inexact graph matching improvements have been proposed. Continuous optimization algorithms [1] is one of the most studied approaches in this field. Typically, methods based on continuous optimization problems are performed in three main steps: (1) transforming the graph matching problem to a continuous problem, (2) using an optimization algorithm to solve the resulting problem and (3) recasting the continuous solution to the initial discrete domain. These approaches can be classified into two broad categories: probabilistic relaxation labeling [67–70] and weighted graph matching problem [71–74].

2.2.3 Graph simulation

Graph pattern matching has been widely used in a broad spectrum of real applications. This problem has been studied with respect to subgraph isomorphism for several applications, such as pattern recognition, dynamic network traffic, knowledge discovery, intelligence analysis, etc [1, 75, 76]. In recent years, several studies have been done to cope with the limitations of the traditional matching paradigm and to catch sensible matches in large graphs. Graph simulation provides an efficient alternative to subgraph isomorphism matching by relaxing some restrictions on matches. The first proposal of graph simulation is in [8], where authors give a quadratic time algorithm for the refinement and verification of reactive systems. Graph pattern matching with graph simulation becomes widely needed in actual applications, such as web site classification, social position detection, plagiarism detection, process calculus, drug trafficking and so on [77–80].

We say that a data graph G matches a pattern graph $Q(V_q, E_q)$, via graph simulation denoted by $Q \trianglelefteq_{sim} G$, if there exists a binary relation $R \subseteq V_q \times V(G)$ which verifies (1) for each $(u, v) \in R$, u and v have the same label, (2) for each query node u, there exists a graph node v such that $(u, v) \in R$, and (3) for each $(u, v) \in R$ and for each edge (u, u') in Q, there is an edge (v, v') in G such that $(u', v') \in R$.

Several studies, such as [81–84], revise the notion of graph simulation and its computation methods for graph pattern matching. In the following, we give an overview of recent studies in revising graph pattern matching.

Authors introduce in [81] bounded simulation model, which extends graph simulation by allowing bounds on the number of hops. In other words, it imposes a weaker structural constraint: (1) it tolerates edge-to-path mapping, *i.e.,* a query edge may be mapped to paths of various bounds in a data graph, and (2) as in graph simulation and in contrast to bijective functions in subgraph isomorphism, bounded simulation seeks a binary relation defined on the nodes of Q and the nodes of G. A variant of the previous proposal, which incorporates regular expressions as edge constraints, is proposed in [82].

In [83], authors present strong simulation model that extends simulation by imposing two additional conditions. The first one is the duality that aims to preserve upward mappings, and the second one is the locality, which helps to eliminate excessive matches.
Dual simulation. We say that a data graph G matches a pattern graph Q, via dual simulation denoted by $Q \trianglelefteq_D G$, if there exists $Q \trianglelefteq_{sim} G$ with a binary match relation $R \subseteq V_q \times V(G)$, and for each pair $(u, v) \in R$ and each (u_2, u) in Q, there exists an edge (v_2, v) in G such that $(u_2, v_2) \in R$. One can see that dual simulation extends graph simulation by preserving both child and parent relationships.
Locality. Authors argue in [85] that the closeness of relationships decreases and the relationships may become irrelevant with the increase of social distance. Keeping potential candidate matches that are in a well-chosen perimeter, bounded by the diameter of the query, can help maintain the meaning of their relationships. Thus,

it often suffices, to some extent, to consider only those matches that fall in a small subgraph.

Both of bounded simulation and strong simulation approaches have a cubic-time complexity, which makes the graph pattern matching process infeasible, especially with massive graphs. Authors highlight in [12] three main approaches to cope with this issue without compromising the accuracy of matches: (1) incremental graph pattern matching approaches, (2) query preserving graph compression and (3) distributed graph pattern matching.

Other approaches addressing the problem of graph pattern matching, based on graph simulation, have been proposed. Based on strong simulation, authors in [86, 87] use graph compression to improve the time of graph querying. These previous works have a cubic complexity in terms of search time. A recent study, given in [84], extends graph simulation by allowing the absence of nodes with one hop. However, it is not a straightforward task to search matches for approximate queries which allow missing nodes and edges. This proposal loses the notion of simulation and loses the quality of matches for queries with leaf nodes. Another recent study is given in [88], which combines a label taxonomy with graph simulation.

2.3 Top-k queries

The top-k query answering problem has drawn a great deal of attention for all data representations, such as relational data, XML and graph. We give in the following a brief overview of the top-k search methods in the literature. We first present some important approaches in the context of relational search. Then we highlight and give a taxonomy of the most important works and methods related to the top-k graph search.

Top-k Relational Queries. In [89], authors give a survey about the top-k query answering in relational database systems. This problem is to find the top-k tuples ranked by a scoring function [89]. The most popular algorithm for top-k querying is the Threshold Algorithm (TA) [90, 91]. Given a monotonic scoring function and lists of data item sorted by their local scores, the authors of [92] introduce an optimal algorithm with a high probability for some monotonic scoring function. This algorithm reads the item value from the lists and builds complete tuples until k complete tuples are found from the top-ranked attributes that have been seen. Then, it conducts random access to find missing scores. In [90], authors improve the previous algorithm by the early termination property and supporting all monotonic scoring function. The main difference is that the access to find missing scores, in the improved algorithm, is guided by predicting the maximum possible score in the unseen ones.

Other algorithms addressing the top-k query answering problem in relational database systems have been proposed. Among them we can cite, ranked join queries over relation data [93–95], ranked join queries over NoSQL databases [96], distance join index [79] and hybrid indexing method [97]. We refer the readers to [89, 98], in which the authors present a survey of top-k query answering problem in database systems.

Top-k graph search. The problem of top-k ranking has also been studied for keyword queries [99, 100], twig queries [14, 101] and subgraph isomorphism [13, 102–105]. It this latter, the problem is about finding the top-k ranking subgraphs that match a query graph, according to given criteria.

The problem of top-k search has attracted much interest on relation data, XML documents and RDF graphs [99, 100, 106–108]. Looking closely, one can find that most of the existing approaches, based on keyword search, use simple ranking functions such as TF/IDF and do not consider the topologies of the returned answers. As against these approaches, the authors of [107] consider, to some extent, the structure of responses. They introduce the *XFinder* system which aims to find the top-k approximate matches of small queries in large documents. This system uses a revised *Prufer sequences* [109] to transform the query and document into strings and the tree edit distance is approximated by the longest subsequence distance between the resulting strings.

Answering top-k queries over XML document is an active research area. The work given in [99] aims to find the top-k subtrees induced from a set of keywords. Authors propose in [110] a top-k approximate subtree matching algorithm based on tree edit distance. We will consider this research field in Section 5.4.2.1 of Chapter 5.

For data graphs, the top-k queries problem is to identify the k subgraphs, that match the query graph, ranked by a score function [14, 104, 111], *e.g.,* the total node similarity scores [104]. Generally, the common practice of most of top-k graph query approaches is the early termination property using a Threshold Algorithm style test. Authors address in [112] the problem of finding best-effort subgraph patterns in attributed graphs. In this study, the algorithm seeks for exact, as well as approximate matches, and it returns them in a defined order. However, this algorithm does not guarantee the k answers with the smallest/largest scores over all answers are returned.

Authors introduce in [113] a neighborhood-based similarity search in graphs, which combines the topological structure and content information together during the search process. This latter is based on a set of rules to identify approximate matches based on their neighborhood structure and labels, where a query graph is also used. Moreover, a similarity function such as graph edit distance is used to measure the similarity of the answer and the query. Another interesting work [114] studies the problem of top-k graph pattern matching with on-the-fly ranked lists based on spanning trees of the cyclic graph query. In this work, the authors propose a multidimensional representation for using multiple ranked lists to answer a given query. Under this representation, they propose a cost model to estimate the least number of tree answers to be consumed in each ranked list. Based on graph simulation, the authors of [115] investigate top–k graph pattern matching. The study presented in [116] deals with the top-k knowledge graph search. In this study, the authors present first a top-k algorithm for start queries. Then they present an assembling algorithm for general graph queries. The assembling algorithm uses start query as a building block and iteratively sweeps the star match lists with dynamically adjusted bound. Recently, authors address in [117] the top-k querying on dynamic graphs.

2.4 Result diversification

Result diversification, which aims to compute the top-k relevant results by considering the *diversity* [118–121], has been widely studied in a large variety of spectrum, such as diversified keyword search in documents [122] and structured databases [123]. Diversity is a general term used to catch the quality of a collection of items with regards to the variety of its constituent elements [121]. Query result diversification can be classified into three main categories [124]. The first category is content-based diversification, also known as similarity-based diversification [15, 125]. Approaches of this category aim to present the dissimilarity between each pair of items. The second category is intent-based diversification, also known as coverage-based diversification [124, 126, 127]. It addresses the user's ambiguous queries where a set of responses covering likely all the different interpretations should be returned. The third category is novelty-based diversification. In this category, objects that contain new information different from ones previously retrieved are privileged in order to improve user's query satisfaction [128, 129]. We encourage readers interested in more details about this classification and the methods within to consult [120] as well as references therein.

In [130], the authors formalize the problem of diversified top-k search. They extend two known algorithms, the incremental top-k algorithm and the bounded top-k algorithm, to solve the diversified top-k search problem by applying three functions, namely, a sufficient stop condition denoted as sufficient(), a necessary stop condition denoted as necessary(), and a diversity search function denoted as div-search-current(). Then, they prove that the div-search-current() is an NP-hard problem and it is hard to be approximated. Thus, they propose three new algorithms, div-astar, div-dp, div-cut, to find the optimal solution for div-search-current().

The study given in [131] focuses on diversification in Keyword search over relational databases. In this study, a formal model is first provided to integrate user preferences into the final ranking. Then, four properties, which are the relevance, the degree of preference of each result, the user interest coverage and the content diversity, are combined to evaluate the quality of returned results. Based on this combined criterion, authors provide efficient algorithms that compute top-k representative results.

The result diversification is a crucial problem in graph querying. In [115], the authors focus on the graph pattern matching using graph simulation by supporting a designated output node. They revise graph pattern matching and then they introduce two functions (relevance and distance functions) to rank matches based on their structure and diversity. In this study, authors provide two algorithms with early termination property, *i.e.*, finding the top-k matches without computing all matches. This latter improves the search time and performs better than the traditional algorithms.

In [132], the authors tackle the problem of extracting redundancy-aware top-k patterns. They first examine two problem formulations: Maximal Average Significance (MAS) and Maximal Marginal Significance (MMS). Then, they present a greedy algorithm that approximates the optimal solution with performance bound $O(log(k))$ for MMS.

Authors study in [133] the problem of top-k diversified subgraph isomorphism that asks for a set of up to k subgraphs isomorphic to a given query, and that covers the largest number of nodes. The authors propose a level-based algorithm for this problem with early termination and an approximation guarantee. A recent study [134] formalizes the top-k shortest paths with the diversity problem.

Other methods for result diversification have been proposed. Among them we can cite, graph feature selection in graph classification [135], node ranking in large graphs [136, 137], diversification in search over unstructured data [127, 138, 139], diversification in querying streaming data [140–143].

2.5 Chapter summary

In this chapter, we tackled the problem of graph querying or graph search. We presented and discussed the state of the art related to graph matching problem and we described the different classes of graph matching algorithms. We focused on the more recent and interesting methods but we also reviewed general approaches. We also presented and discussed the top-k querying and result diversification problems.

In the next chapter, we introduce and present our model for graph pattern matching called relaxed graph simulation, and our algorithms for computing top-k and diversified top-k matches with respect to the proposed model.

Chapter 3

Diversified top-k search with relaxed graph simulation

Contents

Graph pattern matching is being one of the fundamental tasks on which are based the search, the querying, and the analysis of data graphs. This problem has been widely used in a broad spectrum of real-world applications and it has been the subject of several investigations, mainly of its importance and use. In this context, different models along with their appropriate algorithms have been proposed. However, in addition to the excessive processing costs, most of the existing models suffer from the failing query problem due to their limitations on finding meaningful matches. Also, in some scenarios, the number of matches may be enormous, making the inspection a daunting task. In this chapter, we introduce a new model for graph pattern matching, called *Relaxed Graph Simulation (RGS)*, allowing the relaxation of queries to identify more significant matches and to avoid the empty-set answer problem. We then formalize and study the top-k matching problem based on two function classes, relevance and diversity, for ranking the matches with respect to the proposed model. We also formalize and investigate the diversified top-k matching problem, and we propose a diversification function to balance relevance and diversity. Nonetheless, we provide efficient algorithms based on optimization strategies to compute the top-k and the diversified top-k matches according to the RGS model. Our experimental results, on four real datasets, demonstrate both the effectiveness and the efficiency of the proposed approaches.

3.1 Introduction

Graphs are a very useful paradigm for representing and modeling data and their relationships. They consist of a set of nodes representing objects and a set of edges representing relations between these objects. Thus, various real applications, such as protein interactions [6, 144] and social networks [5, 145], use graphs as a model of representation and searching. The flexible nature of graphs allows adding new relationships and even new objects without affecting existing application functionalities and queries. This flexibility brought graphs to greater importance, especially for massive data [146].

In many of these applications, graphs are enormous, with millions of nodes and billions of edges, and it becomes extremely challenging to mine their data. In this context, graph search (or graph querying) is usually used to identify occurrences for a given query. Thus, various types of graph matching have been widely investigated. Graph pattern matching (GPM) is being one of the most widely used operations for a variety of emerging applications. Typically, it can be defined in terms of subgraph isomorphism [7] or graph simulation [8]. Subgraph isomorphism search is the problem of finding all the exact occurrences of a query in the graph. Graph simulation has been adopted for graph pattern matching to cope with restrictions of exact graph matching.

As data graphs are growing in size, the number of matches can be excessively large. Inspecting all the results is a daunting task. Thus, the users are generally interested in responses that satisfy their preferences, which are usually relevance and diversity [15]. In many real-world applications, such as social networks, the matching algorithms use *query focus*, which aims to find matches of a desired node (output node) instead of the entire match [147]. Furthermore, graphs are subject to deformations due to several causes, such as the noise in the acquisition process, the missing or incomplete information and the errors introduced by the modeling processes. So, the obtained results (subgraphs) are likely different from the query graph and its expected answers. That is why the matching process should be able to consider correspondences that do not satisfy, to some extent, all the requirements imposed by the query graph. Besides, the matching process must be able to reach a good approximate solution in an acceptable time, even without a guarantee to give the best solution. These reasons highlight the need for approximate graph matching algorithms. Thus, several approaches have been proposed by extending the traditional graph simulation in order to reach more meaningful matches (see, *eg.*, [12] for a survey). Despite conditions and structural relaxations given by graph simulation and its variants, and since it is almost impossible to know the graph structure, we find that these approaches are restrictive since they do not accept matches with content relaxations, *i.e.,* matches with missing nodes and without affecting the quality of results. In real applications, this kind of relaxation is very useful.

These highlight the need to find the top-k matches of a desired node by allowing relaxation in terms of content (or missing nodes). Let G be a data graph and Q be a pattern graph with the desired node u_*. The top-k graph matching aims to find the k best matches of the desired node u_* in G according to given criteria.

Example 1. A fraction of a collaboration network is shown as G in Figure 3.1b. In this graph, a vertex v_i represents a person with his/her job (label of vertex) and an edge (v_i, v_j) indicates a supervision relationship; *e.g.*, (v_3, v_{12}) indicates that the node v_3 with job A supervises the node v_{12} with job D. A company issues a query to find potential matches and the requirements are given by the query graph Q in Figure 3.1a. Here, u_0 labeled with A and indicated with " $*$ " is the desired node of the query Q. It means that only the matches of this node are asked for.

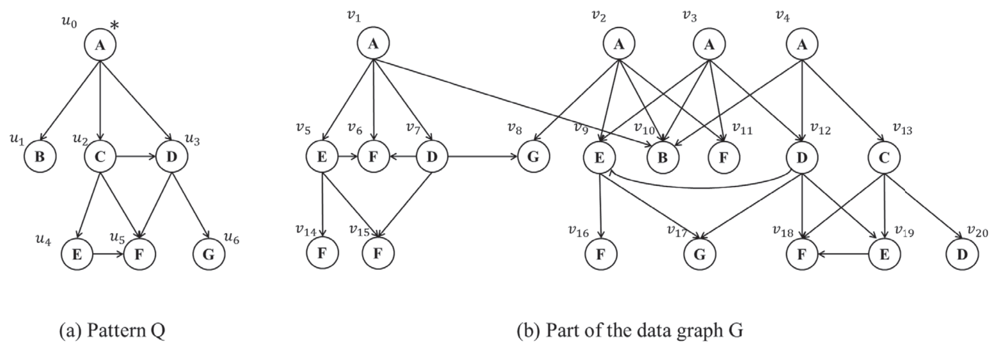


Figure 3.1: A data graph and a query graph

In this example, subgraph isomorphism fails to identify matches for the query Q. Using graph simulation, one can verify that the matching $M(Q, G)$ contains $\{(u_0,v_4)$, (u_1,v_{10}), (u_2,v_{13}), (u_3,v_7), (u_3,v_{12}), (u_4,v_5), (u_4,v_9), (u_4,v_{19}), (u_5,v_6), (u_5,v_{11}), (u_5,v_{14}), (u_5,v_{15}), (u_5,v_{16}), (u_5,v_{18}), (u_6,v_8), $(u_6,v_{17})\}$. The result of this matching includes most of the graph nodes. However, if one looks for matches of the desired node, the result is only the node v_4. Furthermore, one can verify that in such a case, the nodes v_1 and v_3 can be evaluated as deemed matches since it is possible that the supervisor of a person with job C can also be a supervisor of persons supervised by C, *i.e.,* $\{E, F, D\}$. However, traditional graph simulation is often too restrictive to identify this kind of relationship.

Since the user does not have an idea about the content and the structure of the graph, graph search defined in term of graph simulation (including extended graph simulation approaches) may be failing, *e.g.,* the previous case where the relationship (A, C) is missing in the subgraphs induced by v_1 or v_3. In practice, extending graph simulation, by supporting both structural relaxation and content relaxation, provides a good alternative to cope with the drawbacks of traditional matching paradigms. In such a situation, the matching algorithm may benefit from a more general notion of graph simulation, but it may throw away its search efficiency.

Thus, the main contributions in this part of thesis are summarized as follows:

- We propose a new notion of graph simulation, called *Relaxed Graph Simulation (RGS)*, in order to avoid the failing query problem and to reach more significant results.
- We define the graph pattern matching in terms of *RGS*, and we revise this notion to support the query focus by designating a desired node u_*.

- We introduce two classes of functions to rank matches of the desired node u_*. The first one is relevance function $\Omega()$, which measures the relevance of a given match. The second one is distance function $\Theta()$, which measures the dissimilarity of two given matches. Based on both, we define diversification function $F()$, which aims to return diversified answers while maintaining high relevance as much as possible.
- We develop algorithms that calculate the top-k matches and the diversified top-k matches and provide a fast search time. For this, all information around a node are distilled in a probabilistic data structure that makes the search efficient.
- Using real-life data, we conduct extensive experiments to attest the effectiveness and efficiency of the proposed approaches.

The remainder of this chapter is organized as follows. In Section 3.2, we give some definitions that are used throughout this chapter. Section 3.3 is devoted to the description of the relaxed graph simulation. Section 3.4 describes the ranking functions. After that, the search processes are presented in Section 3.5, which are evaluated in Section 3.6. Finally, Section 3.7 concludes this chapter.

3.2 Preliminaries

We investigate the graph pattern matching for directed and unweighted labeled graphs. Without loss of generality, our method could be extended to weighted labeled graphs. In this section, we present some general definitions.

Data graph. Data graphs are used to represent objects and their relationships using nodes and edges. A data graph (or simply a graph) is a directed graph $G = (V(G), E(G), l, \Sigma)$, where (1) $V(G)$ is the set of nodes (also called vertices), (2) $E(G) \subseteq V(G) \times V(G)$ is the set of edges in which (v, v') denotes an edge from v to v', (3) l is a labeling function such that for each node $v \in V(G)$, $l(v)$ is a label from the finite set of labels Σ. We use $|V|$ and $|E|$ to denote the number of nodes and the number of edges in the graph G, respectively. A node v' is a *child* of a node v if $(v, v') \in E(G)$. We use $deg(v)$ to denote the degree of the node v, *i.e.,* the number of children of the node v.

Pattern graph. A pattern graph is a directed vertex-labeled graph $Q = (V_q, E_q, f_v)$, where (1) V_q is the set of query nodes, (2) E_q is the set of query edges, and (3) f_v is a labeling function such that for each node $u \in V_q$, $f_v(u)$ is a label from the set of labels Σ.

Graph pattern matching can be defined in terms of *subgraph isomorphism* or *graph simulation*. In this thesis, we focus on graph simulation, which is defined as follows:

Definition 2. let $G = (V(G), E(G), l, \Sigma)$ be a data graph and $Q = (V_q, E_q, f_v)$ be a pattern graph. The graph G matches the pattern Q via graph simulation, denoted by the matching set $M(Q, G)$, if there exists a binary relation $R \subseteq V_q \times V(G)$ that verifies (1) for each $(u, v) \in R$, $l(u) = f_v(v)$, (2) for each query node $u \in V_q$ there exists a graph node $v \in V(G)$ such that $(u, v) \in R$, and (3) for each $(u, v) \in R$ and for each edge $(u, u') \in E_q$, there is an edge $(v, v') \in E(G)$ such that $(u', v') \in R$.

The authors in [115] give an overview of pattern graphs extended by desired nodes. Based on the study given in [147], they focus on a single desired node. Let $Q^* = (V_q, E_q, f_v, u_*)$ be a pattern graph with the desired node $u_* \in V_q$. The matches of Q^* in G are given by $M(Q^*, G, u_*) = \{v | (u_*, v) \in M(Q, G)\}$.

3.3 Relaxed Graph Simulation

In this section, we introduce and we describe the Relaxed Graph Simulation (RGS) model. We first define the notion of *satisfaction set*, which should be satisfied if two nodes match to each other. A satisfaction set of a query node u, denoted by Sat_u, is a set of label sets that allows checking if a graph node v matches the node u. According to the need in the RGS model (defined after), the satisfaction set for a node u can be computed as follows: (1) the first set in Sat_u contains labels of u's children nodes, (2) for each element in the first set, we replace it by the labels of the corresponding node's children, if they exist, and we produce all possible combinations.

Figure 3.1 shows an example of a part of a data graph (Figure 3.1b) and a pattern graph (Figure 3.1a). One can quickly check the satisfaction set Sat_{u_0} of the desired node u_0. In this example, the first set in Sat_{u_0} contains the children's labels of the node u_0, *i.e.,* $\{B, C, D\}$. According to the above description, the label of the node u_1 will be present in all combinations, since it does not have children. The label C will be replaced by $\{E, F, D\}$ and the label D will be replaced by $\{F, G\}$. The final satisfaction set of the node u_0 is $Sat_{u_0} = \{\{B, C, D\}, \{B, E, F, D\}, \{B, C, F, G\}, \{B, E, F, D, G\}\}$.

Next, we formally define the relaxed graph simulation model.

Definition 3. Let $G = (V(G), E(G), l, \Sigma)$ and $Q = (V_q, E_q, f_v)$ be a data graph and a pattern graph, respectively. The graph G matches the pattern Q via RGS if there exists a binary relation $R \subseteq V_q \times V(G)$ that verifies: if $(u, v) \in R$ then (1) $l(u) = f_v(v)$, and (2) $\exists S_{u_i} \subseteq Sat_u$ such that $S_{u_i} \subseteq L_v$, with L_v is the list of labels of v's children.

From the above definitions, the relation between graph simulation and relaxed graph simulation can be drawn as follows:

Theorem 1. For a given pattern graph and a data graph, we denote the matching set of graph simulation as R_{GS} and that of relaxed graph simulation as R_{RGS}, then $R_{GS} \subseteq R_{RGS}$.

3.4 Ranking Pattern Matches

In general, data graphs are enormous, which makes the result set $M(Q, G)$ excessively large. However, the user's interests can be expressed by the top-k answers of the desired node u_* [89]. This highlights the need for ranking functions to compute the top-k answers for a given query Q.

In this section, we first define the top-k ranking problem and the result-diversity problem. We consider specific functions to measure the relevance and the diversity, respectively. Then, we formalize the diversification problem and we introduce a diversification function, which is a trade-off between the two previous ones.

3.4.1 Top-k ranking for RGS problem

We first define the top-k ranking problem, then we present the scoring function that measures the relevance of the desired node's matches.

Definition 4. Let $G = (V(G), E(G), l, \Sigma)$ be a data graph, $Q^* = (V_q, E_q, f_v, u_*)$ be a pattern graph with the desired node u_*, k be an integer and $\Omega()$ be a score function. A sequence of nodes, $R = (v_{i_1}, v_{i_2}, ..., v_{i_k})$, is *a top-k ranking* of the nodes of the graph with respect to the pattern Q^* iff

1. the ranking contains the k matches that are closest to the desired node u_*: $\forall v_j \notin R : \Omega(v_{i_k}) \geq \Omega(v_j)$, and
2. the nodes in the ranking are sorted by their scores to the desired node: $\Omega(v_{i_j}) \geq \Omega(v_{i_{j+1}})$, with $1 \leq j < k$.

The top-k ranking problem is the problem of computing top-k matches of a query Q^* in a graph G. In the rest of this chapter, we use the notation Q to refer to a pattern with a desired node and $M(Q, G)$ to refer to its match set in G.

Using the *RGS* model, the quality of results depends mainly on the number of reached nodes and the number of substitutions during the matching process.

Reached nodes. Given a graph node $v \in G$ that matches a query node $u \in Q$. Let $U = \{u_1, u_2, \ldots, u_n\}$ be the set of descendants of u in Q. Let $V = \{v_i\}$ be the set of matches in G of every $u_i \in U$. The set of reached nodes $R(u, v)$ is the subset of V containing each vertex v_i such that each vertex of the path u to u_i is in U and each vertex of the path v to v_i is in V. That is, $R(u, v)$ includes all matches v_i reachable by v via a path of matches.

As aforementioned, the *RGS* model is based on the satisfaction set of each node in Q, which is the key to our ranking function. The relation $(u, v) \in R$ means that the v's children set covers at least one set from the u's satisfaction set Sat_u. Each member in Sat_u contains either the labels of the u's children or another list with the substitution of elements in the first list by the labels of the corresponding node's children (as it is described above). This substitution gives the intuition that the importance of a matched node v is according to the corresponding covered set in Sat_u. It is clear that the best match for a node $u \in Q$ is the one that satisfies the first list in Sat_u, *i.e.,* the node that matches the set of direct children. Each substitution affects the importance of the corresponding match and the best quality is the one that corresponds to fewer substitutions.

Let $\alpha \in [0, 1]$ be a penalty factor for each substitution and b_i be the number of substitutions in $S_{u_i} \subseteq Sat_u$ with $0 \leq b_i \leq deg(u)$. We define the score $\gamma(S_{u_i})$ as follows:

$$\gamma(S_{u_i}) = deg(u) - \alpha \cdot b_i. \tag{3.1}$$

Thus, Equation 3.1 gives an overview of the quality of the matched node through the covered satisfaction set. The impact of substitution on the match quality is controlled by α, which is a user-defined parameter. In other words, the penalty factor expresses the influence of the missing nodes on the quality of results. For example, we can define $\alpha = 0$ if we consider that the missing nodes have no impact on the expected results, which is uncommon. As against, if the missing nodes are crucial, then the value of α must be increased, $\alpha = 1$ for example.

We present now the score function (Equation 3.2) for evaluating deemed matches of the desired node. Let the node $v_* \in V(G)$ be a deemed match of the desired node $u_* \in V_q$. Let $v \in V(G)$ be a match of $u \in V_q$ such that v is reached via a path from v_* and u is reached via a path from u_*. The node v covers at least one set from the satisfaction set Sat_u. So for each matched node u, we choose the set $S \subseteq Sat_u$, covered by v, with the best score $\gamma()$. Let S_{best} be the set of the best-covered sets for the matched nodes of the pattern Q, the score $\delta()$ of the node v_* is defined by:

$$\delta(v_*) = \frac{\sum\limits_{S_i \subseteq S_{best}} \gamma(S_i)}{|E_q|}. \tag{3.2}$$

Then, we present the relevance function.

Relevance function. On a match node $v \in G$ of the desired node $u_* \in Q$, the relevance function $\Omega()$ is defined as

$$\Omega(v) = \delta(v) \cdot |R(u_*, v)|. \tag{3.3}$$

Using the relevance function (Equation 3.3), the problem of top-k ranking can be defined as the problem of finding the subset R, of size k, from the set of matches $M(Q,G)$ with the highest similarity among all subsets of size k in M.

$$R = \underset{S \subseteq M, |S|=k}{\arg\max}\ sim(Q, S), \tag{3.4}$$

where

$$sim(Q, S) = \sum_{v_i \in S, i=1}^{k} \Omega(v_i). \tag{3.5}$$

TABLE 3.1: Covered satisfaction sets in G (G given in Figure 3.1)

Match	Covered satisfaction sets
v_1	$\gamma(Sat_{u_0} = \{B, D, E, F\}) = 2.5, \gamma(Sat_{u_3} = \{F, G\}) = 2,$ $\gamma(Sat_{u_4} = \{F\}) = 1, \gamma(Sat_{u_i}(i \in [1, 2, 5, 6]) = \emptyset) = 0$
v_3	$\gamma(Sat_{u_0} = \{B, D, E, F\}) = 2.5, \gamma(Sat_{u_3} = \{F, G\}) = 2,$ $\gamma(Sat_{u_4} = \{F\}) = 1, \gamma(Sat_{u_i}(i \in [1, 2, 5, 6]) = \emptyset) = 0$
v_4	$\gamma(Sat_{u_0} = \{B, C, D\}) = 3, \gamma(Sat_{u_2} = \{E, F, D\}) = 3,$ $\gamma(Sat_{u_3} = \{F, G\}) = 2, \gamma(Sat_{u_4} = \{F\}) = 1,$ $\gamma(Sat_{u_i}(i \in [1, 5, 6]) = \emptyset) = 0$

TABLE 3.2: Reached nodes in G (G given in Figure 3.1)

Match	Reached nodes
v_1	$R(u_0, v_1) = \{v_5, v_6, v_7, v_8, v_{10}, v_{14}, v_{15}\}$
v_3	$R(u_0, v_3) = \{v_9, v_{10}, v_{11}, v_{12}, v_{16}, v_{17}, v_{18}\}$
v_4	$R(u_0, v_4) = \{v_{10}, v_{12}, v_{13}, v_{17}, v_{18}, v_{19}\}$

Hence, the more matches v can be reached, the more significant impact it may have, as observed in [148].

Example 2. Consider G and Q shown in Figure 3.1. One can verify that the nodes of subgraphs induced by the nodes v_1, v_3 and v_4 are mappings for the Q's nodes via *RGS*. Table 3.1 shows the corresponding covered satisfaction sets of the matched nodes in G and Table 3.2 illustrates the corresponding sets of reached nodes. One can verify that $M = \{v_4, v_3\}$ or $M = \{v_4, v_1\}$ is a top-2 ranking according to the *RGS* model. The relevance of the node v_4 is 6, since each node in Q is mapped to at least one node in G, as shown in Figure 3.1. The relevance of the nodes v_3 and v_1 is 4.27 each (with $\alpha = 0.5$). This is due to the fact that the label of the node u_2 is substituted by the labels of its children in Sat_{u_0}.

3.4.2 Match diversity

We next formalize the diversity problem and we introduce a simple metric for result diversity.

Definition 5. Let G be a data graph, Q be a pattern graph, $M(Q, G)$ be the set of matches of the desired node $u_* \in Q$ in G, k be an integer and $div()$ be a dissimilarity function. A sequence of nodes, $R = \{v_{i_1}, v_{i_2}, ..., v_{i_k}\}$ is a set of

k-diverse graph nodes with respect to the pattern Q such that

$$R = \underset{S \subseteq M, |S|=k}{\arg\max}\ div(S), \tag{3.6}$$

where

$$div(S) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \Theta(v_i, v_j). \tag{3.7}$$

The distance function $\Theta(.,.)$ is an essential component of diversification that computes the dissimilarity between the returned matches. In the following, we define a simple distance function based on *Jaccard Coefficient* [149]. Let $v_i \in G$ and $v_j \in G$ be two matches of the desired node u_*, the dissimilarity between v_i and v_j is defined as

$$\Theta(v_i, v_j) = 1 - \frac{|R(u_*, v_i)| \cap |R(u_*, v_j)|}{|R(u_*, v_i)| \cup |R(u_*, v_j)|}. \tag{3.8}$$

The problem of match diversity is to select a subset $R \subseteq M$ of size k that maximizes the sum of inter-element distances amongst elements of R. The above definition is a minor revision of the Max-Sum Dispersion Problem introduced by [150].

Example 3. Given G and Q in Figure 3.1, we observe the following: (1) $div(v_3, v_4) = \frac{5}{9}$; this means that v_3 and v_4 have impact on more than half of the same group of people in G, and (2) $div(v_1, v_3) = \frac{12}{13}$, $div(v_1, v_4) = \frac{11}{12}$. Thus, v_1 and v_3 are most dissimilar to each other.

Next, we formally define the diversified top-k problem.

3.4.3 Diversified top-k matches

Given a list of results $R = \{v_{i_1}, v_{i_2}, ..., v_{i_k}\}$ of size k from the set of matches $M(Q,G)$, the diversified top-k matches, denoted as $D(R)$, is a list of results that satisfy the following conditions:

- $D(R) \subseteq M$ and $|D(R)| \leq k$,
- for any two results $v_i \in R$ and $v_j \in R$ and $i \neq j$, if v_i is similar to v_j, then $\{v_i, v_j\} \nsubseteq D(R)$, and
- the relevance of $D(R)$ is maximized.

The diversified top-k matches are the set of k results, such that the similarity between results is minimized and the sum of relevance is maximized [119]. Based on the above functions, we define the diversification function as follows

$$D(R) = \underset{R \subseteq M, |R|=k}{\arg\max}\ F(R), \tag{3.9}$$

where

$$F(R) = (k-1)(1-\lambda) \cdot sim(Q, R) + 2\lambda \cdot div(R). \tag{3.10}$$

Value λ, with $0 \leq \lambda \leq 1$, is a user-defined parameter that represents the trade-off between similarity and diversity, *i.e.,* it balances between relevance and diversity of matches. The diversity function is scaled up since there are k numbers for the relevance sum and $\frac{k(k-1)}{2}$ for the diversity sum. This function is a minor revision of Max-Sum Diversification introduced by [119].

Example 4. Given G and Q in Figure 3.1, one can verify that $\{v_4, v_1\}$ is a top-2 diversified match set since their diversification value is maximum among all 2-matches of $M(Q, G)$.

3.5 Top-k graph pattern matching algorithms

In this work, we first introduced a new notion of graph pattern matching, called *Relaxed Graph Simulation*, to cope with restrictions of previous approaches and to avoid the empty-set answer problem. Next, we develop algorithms that aim to solve the top-k answering problem and to reduce the cost of the search.

In the following, we proceed with the presentation of two new approaches for the top-k matching problem and the diversified top-k matching problem using the *RGS* model.

3.5.1 Finding top-k matches

The matching process in our approach consists of two main steps. The first step aims to encode the information of each graph node using the *cuckoo filter* [151]. The second step aims to compute the top-k matches of a query Q according to the *RGS* model. The efficiency of the proposed approaches relies on the node

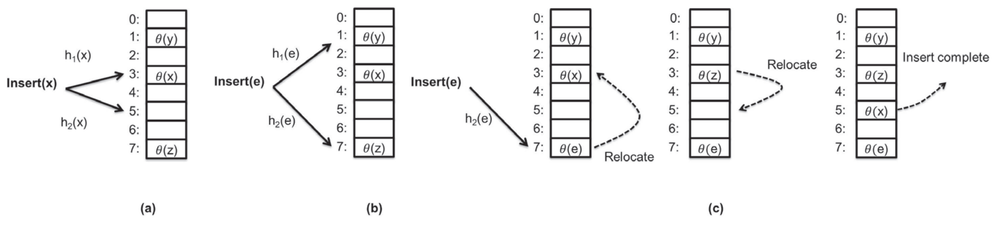


FIGURE 3.2: Illustration of the insertion process in cuckoo filter

encoding, where the *cuckoo filter* is used to distill the neighborhood information into an array of bits. Unlike many existing methods, our encoding can be used for dynamic graphs without additional data structures.

In the following, we first describe the cuckoo filter, and then we detail the matching process.

3.5.1.1 Cuckoo filter

The *cuckoo filter* [151] is a variant of the *cuckoo hash table* [152] that stores a bit string, known as *fingerprint*, obtained by hashing an item instead of a dictionary data structure, *i.e.,* key-value pairs. In this data structure, the basic unit that stores one fingerprint is called *entry* and the hash table is an array of m *buckets* that can have b entries. Two hash functions, $h_1()$ and $h_2()$, are used to identify the buckets for insertion or the lookup.

Cuckoo filter is an efficient probabilistic data structure that supports set membership testing. The authors in [151] show that cuckoo filter is better than *bloom filter* [153]: (a) in terms of lookup performance (runtime performance), (b) it handles deletion, which is not supporting in bloom filter, and (c) it is a space efficiency data structure with low false positive rates ϵ.

In the following, we describe the processes of insertion and lookup in the cuckoo filter.

Insertion. When inserting an item e in the cuckoo filter, one can confront several cases. The first one is when the two buckets in positions $h_1(e)$ and $h_2(e)$ are not allocated, the insertion bucket is chosen arbitrarily. Figure 3.2a shows the insertion of the element x in the bucket 3. The second case is when one bucket is allocated, the insertion is performed in the empty bucket. The last case is when both buckets are allocated. Figure 3.2b illustrates this case. In such a case, the item e can be inserted in one bucket (bucket 7 in this example), and the item that occupies the current bucket will be reinserted in another bucket with reallocation. Figure 3.2c illustrates this case where the item z is inserted in bucket 3 and the occupant of bucket 3 (*i.e.*,, the element x) is inserted in bucket 5.

Inserting a new item in an allocated bucket requires a reallocation for the occupying item. This reallocation requires access to the original items in order to determine their new corresponding buckets. As mentioned before, the cuckoo filter does not store the item itself but it stores its fingerprint $\theta()$. Therefore, it is challenging to find the new bucket of a moved item. To overcome this limitation, the two hash functions are dependent so that they can determine the position of an element based on its fingerprint.

$$h_1(x) = hash(x), \tag{3.11}$$

$$h_2(x) = h_1(x) \oplus hash(\theta(x)). \tag{3.12}$$

Hence, the only need for the reallocation is the information in the table, *i.e.,* there is no need for original items.

Lookup. The lookup using the cuckoo filter is a simple process. Let x be a searched item, the fingerprint $\theta(x)$ is first calculated, then the two bucket positions $h_1(x)$ and $h_2(x)$ are checked. If $\theta(x)$ exists, then the cuckoo filter returns true; otherwise, it returns false.

False Positive Probability. The analyzes of the false positive for the cuckoo filter is detailed in [151]. Let b be the bucket size and $|\theta|$ be the fingerprint size. The false positive rate ϵ depends on b and $|\theta|$, thus the minimal size of the fingerprint is $|\theta| \geq log_2(2b/\epsilon)$ bits.

3.5.1.2 Finding top-k Matches with cuckoo filter

To achieve a fast search time, we need to design powerful algorithms that can benefit from the idea of using the cuckoo filter. Recall that our aim is to find the top-k matches of the desired node u_* according to the RGS model. Let G be a data graph and Q be a pattern graph. We must first identify the set $cand(u_*)$ of candidates of the desired node u_*. From each candidate node, we then check the matching based on the notion of satisfaction set. Using the *RGS* model, we have the following lemmas.

Lemma 1. Given a data graph G and a pattern graph Q, the top-k matches of the desired node u_* can only come from the set of candidates, $cand(u_*)$, of u_*.

Proof. From Definition 4, it is easy to verify that the result of the matching process contains only candidates of the desired node u_*. ■

Lemma 2. Given a data graph G and a pattern graph Q, a graph node $v \in V(G)$ is not a candidate of a query node $u \in V_q$ if $f_v(v) \neq l(u)$.

Proof. From Definition 3, the correspondence of labels is required for a graph node to be a candidate of a query node. ■

Lemma 3. Given a data graph G and a pattern graph Q, a graph node $v \in V(G)$ is not a candidate of a query node $u \in V_q$ if $deg(v) < deg(u)$.

Proof. In Definition 3, the second condition for a graph node v that matches a query node u via the *RGS* model is that one set S_u of u's satisfaction set (Sat_u) is included in the list of labels of the v's children L_v, so $|S_u| \leq |L_v|$. Since $|S_u| \geq deg(u)$ and $|L_v| = deg(v)$, we have $deg(u) \leq deg(v)$. ■

Lemma 4. Given a data graph G, a pattern graph Q and a graph node $v \in V(G)$ that verifies Lemma 2 and Lemma 3 for a query node $u \in V_q$. Let v_{cuckoo} be the cuckoo filter representation of L_v. If $\nexists S_{u_i} \subseteq Sat_u$ that verifies $\forall e \in S_{u_i}$, $\theta(e) \in v_{cuckoo}$, then v is not a candidate of u.

Proof. Assume that $v \in V(G)$ is a candidate of $u \in V_q$ with $S_u \subseteq Sat_u$ is the corresponding satisfaction set, and assume that there is an element $e \in S_u$ such that $\theta(e) \notin v_{cuckoo}$. This means that the element $e \notin L_v$ and therefore $S_u \nsubseteq L_v$, which contradicts Definition 3 (condition 2). ■

We outline Algorithm 1, called *cMatch()*. Typically, the algorithm verifies the last three lemmas to identify deemed matches of the query nodes. (1) It initializes a boolean variable *Match* for the candidacy condition and a set *Sat* to maintain the corresponding satisfaction set if a node v is evaluated as a match of a node u. (2) The algorithm checks if both nodes have the same label (line 3), (3) if so, it checks

Algorithm 1 Cuckoo filter matching cMatch(u,v)

Input: A graph node v and a query node u
Output: Return the satisfaction set $S_u \subseteq Sat_u$ covered by v if v is a candidate for u according to lemmas 2, 3 and 4

```
1: Match := false;
2: Sat := $\emptyset$;
3: if ($f_v(v) = l(u)$) then
4:    if ($deg(u) \leq deg(v)$) then
5:       while (Match = false and $\exists$ unvisited $S_{u_i}$ in $Sat_u$) do
6:          if ($\forall e \in S_{u_i} : \theta(e) \in v_{cuckoo}$) ) then
7:             Match := true;
8:             Sat := $S_{u_i}$;
9:          end if
10:      end while
11:   end if
12: end if
13: return Sat;
```

whether the node v has a sufficient number of children to be a candidate for the node u (line 4). These two investigations reveal if a node satisfies Lemma 2 and Lemma 3. After that, (4) for a node v that satisfies both conditions, the algorithm checks if the list of labels of v's children L_v covers one set of the satisfaction set Sat_u. For each $S_{u_i} \in Sat_u$ and while the match is not yet found, the algorithm checks if L_v covers S_{u_i}. Thus, Algorithm 1 uses the cuckoo filter representation v_{cuckoo} of L_v. For each element $e \in S_{u_i}$ it verifies if the fingerprint $\theta(e)$ belongs to v_{cuckoo}.
Let m be the size of S_{u_i} and n be the size of L_v. In general, the inclusion of S_{u_i} in L_v is checked in $O(n \times m)$ time. Using *cuckoo filter*, the membership of an element e in L_v is checked in constant time and the inclusion of S_{u_i} in L_v is checked in $O(m)$ time, which provides a significant gain for finding matches. (5) Algorithm 1 returns a covered set from Sat_u if it exists; otherwise, it returns an empty set.

Algorithm 2 details the search process. It uses the *cMatch()* subroutine, which verifies Lemma [2-4], and checks whether a graph node is a candidate for a given query node. The *Top-k Cuckoo matching* algorithm first initializes *termination*, which is a boolean variable for the termination condition, and a set R to maintain matches of the desired node u_* ranked by their relevances $\Omega()$ (lines 1-2). The algorithm selects, from the candidate set $cand(u_*)$ of u_*, an unvisited node and invokes *CheckMatch()* subroutine to check whether this node is a *veritable match* (lines 4-14). If a candidate of u_* is evaluated as a veritable match, Algorithm 2 invokes *CheckAdd()*, which evaluates the score of the node $\Omega()$ and performs the

Algorithm 2 Top-kCuckoo matching

Input: A graph G, a pattern Q^* and a positive integer k.
Output: R: a top-k match set of the desired node u_*.

```
 1: termination := false;
 2: R := $\emptyset$;
 3: cand($u_*$) := {non matched $v \in V(G)$ such that $|\text{cMatch}(v, u_*)| \neq 0$};
 4: while (termination = false and $|cand(u_*)| > 0$) do
 5:     Choose a node $w \in \text{cand}(u_*)$;
 6:     if ($w \neq \emptyset$) then
 7:         if (CheckMatch($Q, w$) = true) then
 8:             CheckAdd($w, R$);
 9:             check and update the termination condition;
10:         end if
11:     else
12:         termination := false;
13:     end if
14: end while
15: return R;
```

insertion in the list of answers R. If this node has a score $\Omega()$ better than the score of the last element in R, then it must be inserted in R (line 8). Algorithm 2 checks the termination condition: if R has k elements with the highest score, the variable *termination* will be updated with false (line 9). If this last holds or the set of candidates is empty, Algorithm 2 returns R as a response to the query.

Subroutine CheckMatch(). The starting point in *CheckMatch()* is the satisfaction set covered by the list of children of the current candidate node. *CheckMatch()* looks for mapping each element in the current satisfaction set against the children of the corresponding graph node. If at least one element is not mapped (according to the *RGS* model), the corresponding candidate of u_* is evaluated as a non-match, and the process tackles the next candidate. Otherwise, each mapped query node can have one of the two following states: *(a)* the mapped query node has no satisfaction set or *(b)* the mapped query node has a satisfaction set. The first state is simple; no further processing is required for this node. For the second state, *CheckMatch()* looks for mapping the elements of the children list of this node, *i.e.,* the elements of the corresponding satisfaction set, and it performs the same previous evaluation for each generated satisfaction set until all investigations for the mapping are performed.

Example 5. Consider data graph G and query graph Q given in Figure 3.1. When Q is issued on G, *top-kCuckoo matching* identifies the top-2 matches as follows: Firstly, v_1, v_3 and v_4 are selected as candidate matches of the query node u_0. Suppose that the node v_1 is evaluated first, the corresponding satisfaction set covered by v_1 is $\{B, D, E, F\}$ with $\gamma = 2.5$. Algorithm 2 looks for mapping each corresponding node with a label in the satisfaction set to a graph node, the node u_1 is mapped to v_{10} with $\gamma = 0$, the node u_3 is mapped to v_7 with $\gamma = 2$ since the list of v_7's children covers the u_3's satisfaction set $\{F, G\}$. The node u_4 is mapped to v_5 with $\gamma = 1$ and the u_4's covered satisfaction set is $\{F\}$. The last corresponding node in the u_0's satisfaction set u_5 is mapped to v_6 with $\gamma = 0$. After this first evaluation of elements of the first satisfaction set, Algorithm 2 evaluates elements of the resulting satisfaction sets in the same manner. The corresponding nodes of F, G are mapped to v_{15} (or v_6), v_8 respectively, the node u_5 (from the u_4's satisfaction set) is mapped to v_{15} or v_6. At this end, all the evaluations are performed and the node v_2 must be added to R with the score $\Omega(v_1) = 4.27$. Similarly, the nodes v_3 and v_4 are evaluated and their final scores are $\Omega(v_3) = 4.27$ and $\Omega(v_4) = 6$. Hence, *Top-k Cuckoo matching* returns $R = \{v_4, v_1\}$.

Complexity. We give an analysis of the running time of the proposed approach. Let D and d be the average degrees of the graphs G and Q respectively. The satisfaction set Sat_u of a node u contains 2^{d_u} sets S_{u_i} each of size at most ${d_u}^2$, with $d_u = deg(u)$. For a given node v with a degree d_v, the cuckoo filter construction takes $O(d_v.h.m.b)$, where h represents the number of the used hash functions, m is the number of buckets and b is the number of entries for one bucket. Since $h = 2$ and $m.b \equiv O(D)$, the construction of this filter takes $O(|V|.D)$ time for the entire graph G. In the cuckoo filter, each element of S_{u_i} can be checked in $O(1)$, so the checking process of S_{u_i} takes $O(d^2)$ time. As all $S_{u_i} \subseteq Sat_u$ should be checked, the total time of *cMatch()* for a given node u is $O(2^d.d^2) \equiv O(2^d)$. *CheckMatch()* subroutine uses *cMatch()* to check whether a node $v \in G$ is a candidate for the desired node u_0 (in time $O(2^d)$). This verification for all the nodes of Q is then done in $O(|V_q|.2^d)$ time. Thus, to check all nodes of G we need $O(|V|.|V_q|.2^d)$ time. Finally, the *CheckAdd()* uses a min-heap to maintains the results in $O(|V|.log\, k)$. Thus, the *top-k Cuckoo* is done in $O(|V|.D + |V||V_q|.2^d + |V|.log\, k) \equiv O(|V|.D + |V||V_q|.2^d)$ time.

In graph pattern matching the size of a pattern Q is typically small and the maximum degree in this pattern is often small too [12]. Thus, since the time complexity of our approach depends on the average degree d of the query Q and d is generally small (related to the query size), we have $|V_q|.2^d << |V| + |E|$ and the complexity of *top-k Cuckoo* becomes $O(|V|.D + |V|.(|V| + |E|))$, which gives

Algorithm 3 Top-kDiv

Input: A data graph G, a pattern Q^*, a positive integer k and $\lambda \in [0-1]$.
Output: R: a diversified match set R of the desired node u_*, $|R| = k$.

```
1: R := $\emptyset$;
2: Compute $M(Q, G, u_*)$;
3: while ($|R| < k$) do
4:     $x := \arg\max_{x \in M} F(x, R)$;
5:     $R := R \cup \{x\}$;
6:     $M := M - \{x\}$;
7: end while
8: return R;
```

a better time complexity than *top-kGS* (given by $O((|V_q| + |E_q|)(|V| + |E|) + |V|.(|V| + |E|))$ [115]).

3.5.2 Finding diversified top-k matches

In the following, we proceed with the presentation of the diversified top-k matching problem. In contrast to the top-k matching problem, which is based only on the $sim()$, the diversified top-k problem is intractable.

Theorem 2. The diversified top-k problem is NP*-complete (decision problem).*

Proof. Given a graph G, a pattern Q, an integer k, $\lambda \in [0-1]$ and a bound B, the decision problem of Top-kDiv (Algorithm 3) is to decide whether a k-element set $R \subseteq M(Q, G)$ with $F(R) \geq B$ exists. In the following, we show the proof given in [115]. Since we can guess a k-element set R and then check whether $R \subseteq M(Q, G)$ and $F(R) \geq B$ in PTIME, so the decision problem of diversified top-k is in NP. Furthermore, the diversified top-k problem is NP-*hard* since it is a general case of the k-diverse set problem [154], which is known to be NP-*hard*. Thus, the diversified top-k problem is NP-*complete*. ■

Despite the hardness, we provide an approximation algorithm for the diversified top-k problem. Intuitively, this problem can be decomposed into two processes. The first one is the graph pattern matching, which computes the match set $M(Q, G)$, and the second one is the result diversification.

Algorithm. Given a graph G, a pattern Q, an integer k and $\lambda \in [0-1]$, Algorithm 3, denoted as *Top-kDiv*, identifies a set of k matches of the desired node u_* that maximize $F()$. (1) It initializes a set R to maintain the matches of the desired node u_*. (2) It computes $M(Q,G)$ with the same strategy used in Algorithm 2 with respect to the RGS model. (3) *Top-kDiv* incrementally builds the result set R by selecting a pair of matches that maximizes the diversification value $F()$. It then performs the insertion of elements in R and removes them from $M(Q,G)$. Finally, Algorithm 3 returns R as a response to the pattern Q.

Example 6. Consider data graph G and pattern graph Q given in Figure 3.1, and assume that $\lambda = 0.5$ and $k = 2$. When Q is issued on G, *Top-kDiv* identifies the top-2 diversified matches of the query Q as follows: Firstly, $\{v_1, v_3, v_4\}$ are selected as matches of the desired node u_*, *w.r.t.* *RGS*. Then a pair of matches that maximizes $F()$ is selected. Hence the pair $\{v_4, v_1\}$ is selected and returned as the final result, since the diversification value $F(v_4, v_1) = 6.05$ is maximized.

3.6 Experiments

In this section, we describe and discuss experimental results to evaluate our methods. We evaluate the performances of the proposed approaches over various types of real graphs, number of labels and sizes of queries.

3.6.1 Experimental setting

All experiments are conducted on a machine with 3.19 Ghz Intel Core i7 CPU and with 16 GB of RAM running Windows 7. All algorithms are implemented in Java. In the following, we present the test collections and the evaluation method then we present our results.

3.6.1.1 Datasets

For the experiments, we use four datasets of real-world graphs from the Stanford Large Network Dataset Collection [1].

- *Epinions.* This dataset consists of a graph representing who-trust-whom online social network. This graph contains $75,879$ nodes, $508,837$ edges and 50 synthetic labels with a uniform random distribution on the nodes.

[1]http://snap.stanford.edu/

Table 3.3: Graph dataset characteristics

Dataset	$\|V\|$	$\|E\|$	Number of labels
Epinions	$75,879$	$508,837$	50
Amazon	$400,727$	$3,200,440$	100
Google	$875,712$	$5,105,039$	200
LiveJournal	$4,847,571$	$68,993,773$	500

- *Amazon.* This is a real co-purchasing network with $400,727$ nodes, $3,200,440$ edges and 100 synthetic labels with a uniform random distribution on the nodes.

- *Google.* In this graph, nodes represent web pages and directed edges represent hyper-links between them. This dataset contains $875,713$ nodes, $5,105,039$ edges and 200 synthetic labels with a uniform random distribution on the nodes.

- *LiveJournal.* This dataset is a graph representing an on-line social network with almost 5 million members ($4,847,571$) and over 68 million edges ($68,993,773$), which represent friendship relations. We use 500 distinct labels with a uniform random distribution on the nodes.

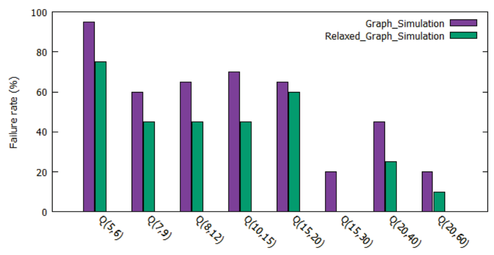
The characteristics of the datasets are summarized in Table 3.3. For each graph, we report the number of nodes, the number of edges and the number of unique labels.

3.6.1.2 Pattern generating

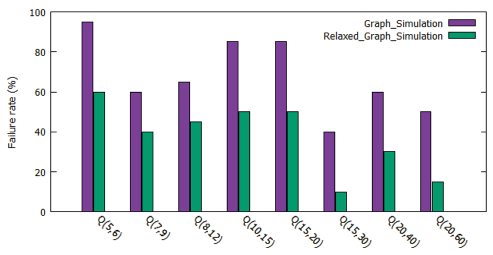
For the experiments, we generate several queries with different sizes, controlled by three parameters: number of nodes, number of edges and labels from the list of graph labels Σ. We use $(|V_q|, |E_q|)$ to denote the query size, and for each size, we generate 20 different query graphs.

3.6.1.3 Evaluation method

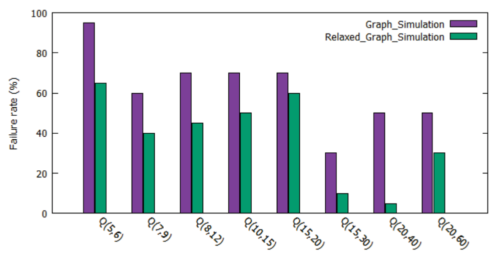
We use two criteria to evaluate the experiment results. The first one is the *failure rate*, which represents the ratio between the number of failing queries, *i.e.,* for which the algorithm cannot find a response, and the number of queries used.



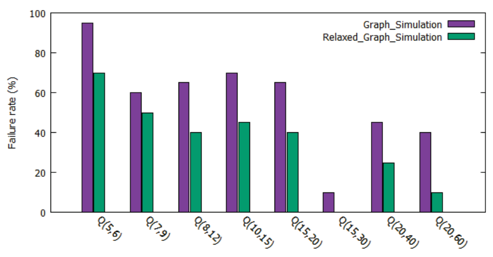
(A) Epinions



(B) Amazon

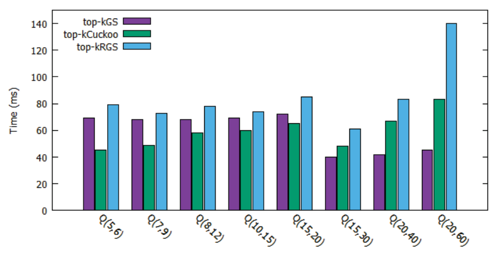


(C) Google

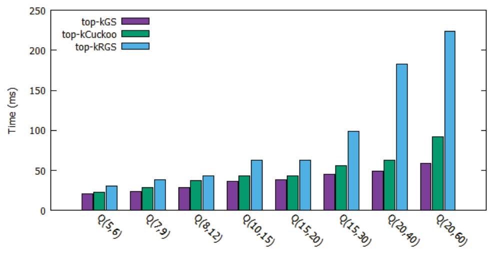


(D) LiveJournal

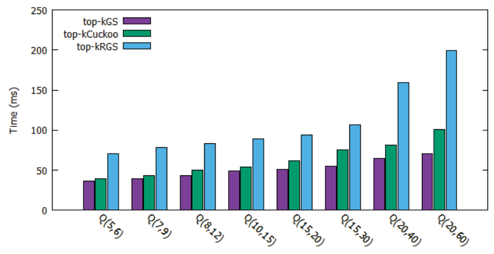
Figure 3.3: Failure rate of Relaxed Graph Simulation and Graph Simulation



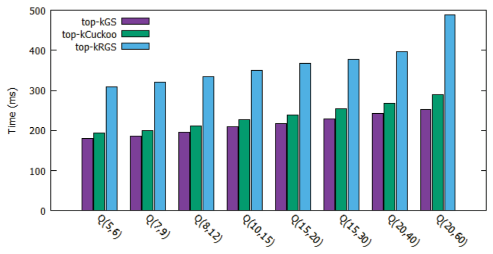
(A) Epinions



(B) Amazon

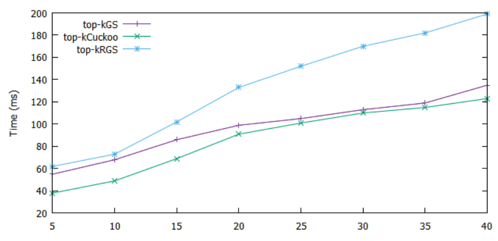


(C) Google

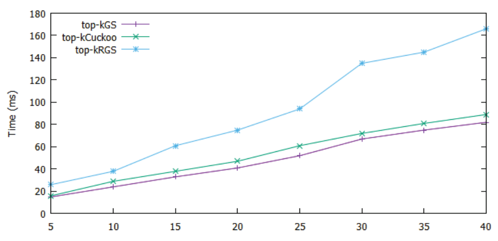


(D) LiveJournal

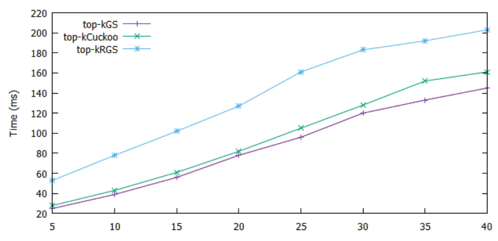
Figure 3.4: Search time for the top-k search problem (Varying $|Q|$)



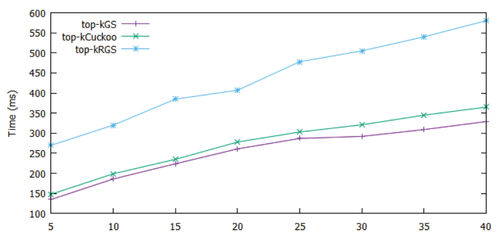
(A) Epinions



(B) Amazon

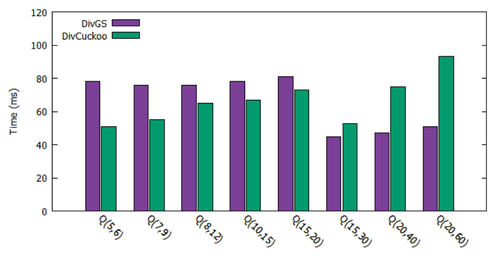


(C) Google

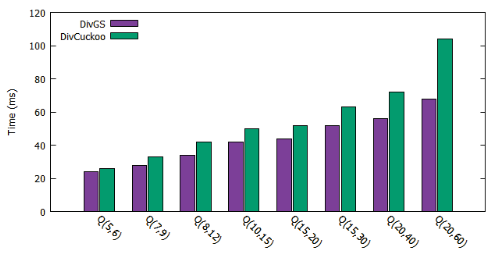


(D) LiveJournal

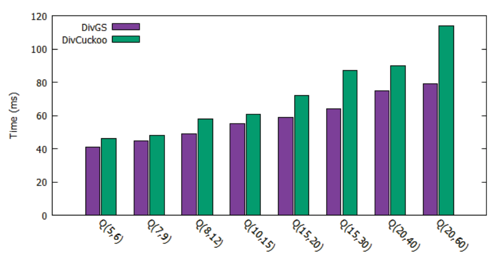
FIGURE 3.5: Search time for the top-k search problem (Varying k)



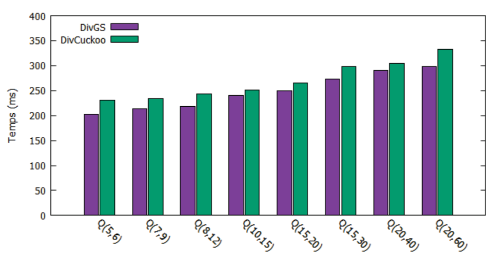
(A) Epinions



(B) Amazon



(C) Google



(D) LiveJournal

FIGURE 3.6: Search time for the diversified top-k search problem (Varying $|Q|$)

This criterion evaluates the effectiveness of algorithms. The second criterion is *efficiency* in terms of search time. In the following, we demonstrate the usefulness of our propositions by conducting three sets of experiments.

The first experiment, called *failure rate*, analyzes the effectiveness of the *RGS* model against the *GS* model using the four datasets.

The second experiment, called *top-k search*, analyzes the efficiency of the proposed algorithms for the top-k matching problem. Three sets of algorithms are investigated: (1) *Top-kGS* is the implementation of the top-k search algorithm using the *GS* model and the early termination property [115]. This algorithm is well evaluated in the literature. It takes on average only 60% of the time of the same process without the early termination property. (2) *Top-kRGS* is the implementation of the top-k search algorithm using the *RGS* model. (3) *Top-kCuckoo* is the implementation of the top-k search algorithm using the *RGS* model and the cuckoo filter as an optimization strategy (Algorithm 2). Although the *RGS* and the *GS* models are different, we compare the RGS-based algorithm with the GS-based one to prove the effectiveness and the efficiency of the proposed approach.

The last experiment, called *Diversified top-k search*, evaluates the efficiency of the proposed diversified top-k search algorithm. We compare our algorithm (called *DivCuckoo*), which is the implementation of the diversified top-k algorithm using the *RGS* model and the cuckoo filter, with the diversified top-k algorithm (called *DivGS*), which is the implementation of the diversified top-k search algorithm using the *GS* model and the early termination property [115].

3.6.2 Results

In the following, we report and discuss the obtained results.

3.6.2.1 Failure rate

We first evaluate the effectiveness of the *RGS* model compared to the *GS* model. Figures 3.3a, 3.3b, 3.3c and 3.3d report the results of the failure rate of the two searching models on Epinions, Amazon, Google and LiveJournal datasets, respectively. In this experiment, we set $k = 10$ and we varied the query size from $(5, 6)$ to $(20, 60)$. We observe that the *RGS* model effectively reduces the failure rate in all experiments. For instance, the RGS's failure rate using $Q(5,6)$ is $20\%, 35\%, 30\%$ and 25% less than the GS's failure rate on Epinions, Amazon, Google and Live-Journal, respectively. We observe almost the same results for all other queries on all datasets.

Case study. The results presented in the figures of failure rate show whether the algorithm returns at least one response. Nevertheless, this does not help to know if the algorithm has identified enough results, *i.e.,* whether the number of returned results is close to the desired k. For all datasets, the result lists of both models are inspected, and we observe that (1) the returned results correspond to the desired queries and (2) the RGS model surpasses the GS model. For instance, we kept $k = 10$ and we consider queries of size $Q(8,12)$ on the Amazon dataset. These queries correspond to the minimum difference between the failure rates of the two models (the GS's failure rate is 65% and the RGS's failure rate is 45%). For the queries that have been answered by the GS-based algorithm, this latter returned $1, 3, 3$ and 5 answers for 4 different queries. However, the RGS-based algorithm was able to find $7, 10, 9$ and 10 answers for the same queries. These observations are for small queries and the problem persists for larger ones.

3.6.2.2 Top-k search

The *search time* criterion shows the efficiency of the proposed approach. In what follows, we report the search time of the three algorithms. We first start with the search time by varying the query size, then we present the search time by varying the number of desired answers k.

Varying query size. Figures 3.4a, 3.4b, 3.4c and 3.4d show the average search time per query size (20 queries per size) after running 30 times in the same setting as the *Failure rate* experiment. The results show that *top-kGS* and *top-kCuckoo* always outperform *top-kRGS*. On Epinions dataset, *top-kCuckoo* takes less search time than *top-kGS* for small queries, but it takes more search time for large queries, which is entirely reasonable since *top-kCuckoo* identifies more matches. Generally, *top-kGS* and *top-kCuckoo* have the same time on average. *top-kCuckoo* takes only 61% of the time of *top-kRGS* on average. On Amazon dataset, *top-kGS* takes 78% of the time of *top-kCuckoo*, which outperforms *top-kRGS* by 49% on average. On

Google dataset, *top-kGS* takes 82% of the time of *top-kCuckoo* on average. On the other hand, *top-kCuckoo* improves *top-kRGS* by 48%. On LiveJournal dataset, *top-kGS* takes 89% of the time of *top-kCukcoo*, which improves *top-kRGS* by 36% on average.

Varying k. For this experiment, we have varied k from 5 to 40 with an incrementation of 5, and we have chosen the query size $Q(7,9)$ since it has the minimum difference between the failure rate of the two models. Figures 3.5a, 3.5b, 3.5c and 3.5d report the search time by varying k. The three algorithms are sensitive to k. The search times of *top-kGS* and *top-kCuckoo* are close, and both of them outperform *top-kRGS*.

In terms of scalability, both models scale well with the graph size, which is approved by the experiments on Google and LiveJournal datasets. The search times of *top-kCuckoo* and *top-kGS* are less sensitive to the variation of k than those of *top-kRGS*.

3.6.2.3 Diversified top-k search

Finally, we evaluate the efficiency of *DivCuckoo* against the *DivGS*. In this experiment, we do not report the results of the diversified top-k search without using the Cuckoo filter optimization since it has a longer search time compared to the two other algorithms (as shown in the previous experiments). We keep the same setting as the *Varying query size* experiments (Section 3.6.2.2) and we set $\lambda = 0.5$.

Figures 3.6a, 3.6b, 3.6c and 3.6d show the average search time per query size (20 queries per size) after running 30 times. These results are consistent with the previous experiments. On Epinions dataset, *DivCuckoo* outperforms *DivGS* for small queries, which is unexpected since it performs more test and it identifies more matches. Furthermore, *DivGS* takes less search time for large queries, which is quite reasonable according to the principle of each model (*GS* and *RGS*). On the other datasets, we observe that *DivGS* and *DivCuckoo* have an almost similar search time on average for almost small queries. We also observe the same results as those on Epinions dataset for large queries.

Both algorithms are sensitive to the query size ($|V_q|, |E_q|$). However, *DivCuckoo* is more sensitive due to the extra time incurred by larger queries to compute the matching using the *RGS* model.

Case study. To evaluate the quality of our algorithm, we manually inspected the top-3 diversified matches. We observed that the returned results correspond to the queries and the *DivCuckoo* substituted at least one intermediate result by another one to diversify the final results.

These experiments show the performance of the proposed approaches, using the *RGS* model, in terms of quality and with an almost similar search time to those using the traditional *GS* model.

3.7 Chapter summary

In this chapter, we have addressed the (diversified) top-k graph pattern matching problem. We have introduced and studied a new model of graph pattern matching called Relaxed Graph Simulation (*RGS*). Based on the *RGS* model, the search process can achieve more meaningful matches and avoid the empty-set answer problem by considering matches with missing nodes and without affecting the quality of results. It provides good flexibility for several applications, such as social networks. We have also defined functions to measure the relevance and diversity of the returned matches. Based on both of them, we have proposed a diversification function that balances the two criteria. Besides, we have developed efficient algorithms using the cuckoo filter for computing the (diversified) top-k matches. Therefore, our approach is very suitable for large graphs due to its scalability. The experiments validate the effectiveness and efficiency of this approach.

Chapter 4

Aggregated search: Definition and overview

Contents

Information Retrieval (*IR*) encounters a migration from the traditional paradigm, which aims to return a list of ranked responses, to the aggregated search paradigm, which aims to provide integrated search services that bring the most focus and relevant answers to the user's queries. In other words, in response to a user's query, traditional information retrieval systems back a ranked list of links that reference potential answers examined by the user to find those that likely fulfill his/her need. For some queries, the user could find the link that satisfies his/her need (*e.g.* the date of the second world war). However, for other kinds of queries (*e.g.* the second world war), one link (or document) is not enough since the relevant information may be scattered across several documents. In such a case, the user should collect and aggregate all relevant and non-redundant information, from different data sources, which can respond to his/her expectations. Querying all these data sources and combining the returned results in the same aggregate to achieve better relevance and better organization is the scope of the aggregated search.

In this chapter, we firstly present the aggregated search paradigm in the field of Information Retrieval (*IR*), which is one of the most exciting problems in *IR*. We also survey and discuss related *IR* disciplines and tools.

4.1 Aggregated search

Recently, information retrieval systems have been expanding to address more range of information-seeking tasks. Examples include search according to the type of information (*e.g., image, document, video...*) or according to the domain of search (*e.g., shopping, travel, news ...*). Each information-seeking task requires, in most cases, a customized solution. In other words, different tasks may require different representations of information, different retrieval algorithms and different ways of returned-results representation. For examples, (1) images and books require two different representations, *e.g.,* images can be represented by text from the surrounding context in the originating page [155] and books can be represented by text from an external summary page [156]. (2) Geographic-based search and news search require two different retrieval algorithms, *e.g.,* local business search may require favoring businesses that are geographically close [157], news search may require favoring recently published articles [158]. (3) A video search and web-page search require two different representations, *e.g.,* videos are displayed using a still-frame of video, a description and a duration while web-page results are displayed using the title and a summary showing the context where the query terms appear [159].

The diversity of user needs makes information retrieval engines more and more specialized. Typically, an engine that supports multiple search tasks uses a specialized system for each supported task. How can such an approach be achieved by integrating search across these widely different systems? This is the goal of the aggregated search. Aggregated search addresses the task of searching and assembling information from a variety of data sources and placing them in a unified interface or document. Using this search paradigm makes it possible to exploit a wide range of functionalities and advantages in order to obtain better results, in terms of precision and quality, to a user query.

4.1.1 Motivation

Typically, Information Retrieval (IR) engines return, in response to a user query, a list of ranked documents or links that are evaluated as deemed matches. These returned documents are ranked by criteria functions that deal with features of documents and queries. The user should go through this response list to examine each returned document, by starting with the top one, in order to find relevant answers to his/her query. As a result of current information features, the returned list may become huge, and the relevant information may not be a single adjoining document which makes these systems outdated concerning the satisfaction of the user needs.

Most of the used matching processes in these engines focus on relevant information at the document level. Each of them is based on a theoretical model such that the probabilistic model [160], vector space model [161] and language model [162].

This vision of the document-level ranking is limited and may not be appropriate to meet user expectations [19–21] for several reasons. In the following we cite some limitations: (1) a relevant information may not be an entire document, it may be contiguous or scattered sections (information) in the same document as it may be scattered across several documents, (2) the way in which returned results are represented, often as a ranked list, is not always appropriate to the user's expectations above all when the answer is just a part of document, (3) the interpretation of queries differs from one context to another, which makes them ambiguous in term of information need. The example in such a case is when a term refers to several things at once, especially when different sources of information are used to construct the answer.

These mentioned limitations are just some of many other limitations of traditional information retrieval systems. Thus, there is a need for more focus, more organization, and more diversity in the returned results. Besides, more in-depth search methods, which process information with finer granularity and build a response by combining multiple contents that may be useful to the user, are needed. Aggregated search addresses these tasks of searching and assembling results.

4.1.2 Definition

Google[1] was the first who explicitly introduced the idea of aggregated search as universal search:

”Google’s vision for universal search is to ultimately search across all its content sources, compare and rank all the information in real time, and deliver a single, integrated set of search results that offers users precisely what they are looking for. Beginning today, the company will incorporate information from a variety of previously separate sources – including videos, images, news, maps, books, and websites – into a single set of results. At first, universal search results may be subtle. Over time users will recognize additional types of content integrated into their search results as the company advances toward delivering a truly comprehensive search experience.”

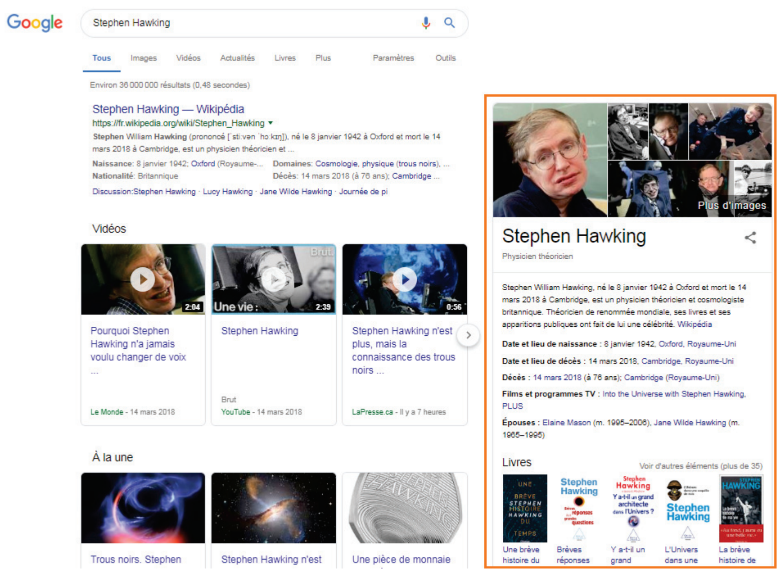
Moreover, the first definition of the aggregated search was given at the ACM SIGIR 2008 Workshop on Aggregate Research [20]:

Definition 6. Aggregated search is the task of searching and assembling information from a variety of sources, placing it into a single interface.

In other words, the goal of the aggregated search is to provide integrated search across multiple heterogeneous sources and construct an aggregate answer, that contains the most relevant, exhaustive and non-redundant information, to be returned in a unified interface (document) and a common presentation of results.

For instance, the provided results for the query “Stephen Hawking” in Google search engine[2] (see Figure 4.1), in the red rectangle that represents the aggregated search result, contain a short biography, images, associated books, and movies.

[1]http://googlepress.blogspot.com/2007/05/google-begins-move-to-universal-search_16.html
[2]March 2019



Environ 36 000 000 résultats (0,48 secondes)

Stephen Hawking — Wikipédia
https://fr.wikipedia.org/wiki/Stephen_Hawking
Stephen William **Hawking** (prononcé [ˈstiːvən ˈhɔːkɪŋ]), né le 8 janvier 1942 à Oxford et mort le 14 mars 2018 à Cambridge, est un physicien théoricien et ...
Naissance: 8 janvier 1942; Oxford (Royaume-... **Domaines**: Cosmologie, physique (trous noirs), ...
Nationalité: Britannique **Décès**: 14 mars 2018 (à 76 ans); Cambridge ...
Discussion:Stephen Hawking · Lucy Hawking · Jane Wilde Hawking · Journée de pi

Vidéos

Pourquoi Stephen Hawking n'a jamais voulu changer de voix ...
Le Monde - 14 mars 2018

Stephen Hawking
Brut
YouTube - 14 mars 2018

Stephen Hawking n'est plus, mais la connaissance des trous noirs ...
LaPresse.ca - Il y a 7 heures

À la une

Trous noirs. Stephen
Stephen Hawking n'est
Une pièce de monnaie

Stephen Hawking
Physicien théoricien

Stephen William Hawking, né le 8 janvier 1942 à Oxford et mort le 14 mars 2018 à Cambridge, est un physicien théoricien et cosmologiste britannique. Théoricien de renommée mondiale, ses livres et ses apparitions publiques ont fait de lui une célébrité. Wikipédia

Date et lieu de naissance : 8 janvier 1942, Oxford, Royaume-Uni
Date et lieu de décès : 14 mars 2018, Cambridge, Royaume-Uni
Décès : 14 mars 2018 (à 76 ans); Cambridge (Royaume-Uni)
Films et programmes TV : Into the Universe with Stephen Hawking, PLUS
Épouses : Elaine Mason (m. 1995–2006), Jane Wilde Hawking (m. 1965–1995)

Livres
Voir d'autres éléments (plus de 35)

Une brève histoire du
Brèves réponses
Y a-t-il un grand
L'Univers dans une
La brève histoire de

FIGURE 4.1: Aggregated search result in Google web search engine

This is why this kind of search engine uses several techniques that aim to search and provide results from various sources. Then it integrates diverse contents, provides pertinent answers and assemble related contents in the same frame.

Based on the above, aggregated search can be seen as a mature subfield of previous works in the information retrieval fields such as federated search, metasearch, semantic search, natural language generation and so on, which will be discussed later from the perspective of aggregated search. Typically, most aggregated search systems follow an architecture with three main sub-tasks [21]: (1) *Query dispatching*, concerned with how to analysis the used query and how to select the information sources, (2) *fragment retrieval*, concerned how to select fragments that contain pertinent information, from which relevant documents can be retrieved, and (3) *result aggregation*, concerned with how to assemble results from the retrieved fragments so as to best represent the final result with the most relevant information. Figure 4.2 shows a general aggregated search framework that uses the three sub-tasks. In the following, we will discuss them in more detail.

In the query dispatching step, we include the actions that precede query matching, that is, initial interpretation of the query (query analysis) and other actions that depend mainly on the query and knowledge about the collections. We can also see this step as deciding which solutions should be triggered for a given query. We distinguish between the approaches that aim at selecting the right sources to be used, the approaches that try to understand more about the query, and the

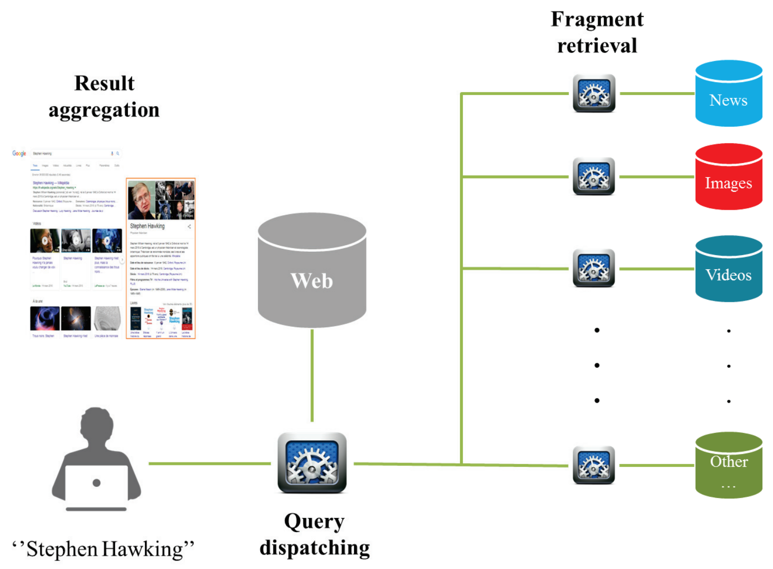


Figure 4.2: Aggregated search framework

approaches that try to extend the query to have more chances to find suitable search results. We will list here briefly well-known problems in this context:

- **Query dispatching:** the first sub-task deals with two operands before processing the query matching. The first operand is the query where each submitted one undergoes an analysis operation that aims to interpret and reformulate the query. The second operand is the information sources.
  - Query analysis: this action aims to interpret and reformulate the query in order to decide which solutions should be triggered. We distinguish two main operations: (1) *Query interpretation* that aims to discover the intent of the question, to find what type of question is being asked (*e.g.* wh-questions, yes/no questions, ...) and to identify semantic relations between the query and the answers [163]. In this respect, several solutions are envisaged, such as query decomposition in order to make the retrieval process easier [164, 165]. (2) *Query reformulation* aims to add some features, to adapt and to personalize the query. These operations are mainly based on the interpretation of the query and the sources to be interrogated. To illustrate this concept, one can consider a search engine that interrogates XML collections. Usually, the query is in a textual format which should be translated to an appropriate format (*e.g., XML fragment*) that allows considering content and structure using the same query [166].

  - Source selection: is to predict which sources (if any) seem relevant to the query. One can view this task as that of deciding which sources should be interrogated and which type of information should be displayed on the aggregated search results regardless of their position. Furthermore, it is impractical to issue the query to every source of information [159]. For this reason, most approaches use pre-retrieval evidence that is based on particular keywords; for example, the query is related to a specific domain such as health domain, contains the term "news" or the query contains the name of a location, etc.

- **Fragment retrieval:** is the sub-task that follows the query dispatching and precedes the result aggregation. That is, it aims to extract information fragments, that seem relevant to the query, and return them to construct the final answer that will be sent to the next sub-task. Typically, on the one hand, this sub-task depends on the nature and the type of interrogated sources, and on the other hand, the type of expected results. To this end, there are many types of fragment retrieval approaches that can be used in the aggregated search process. This includes textual retrieval, image retrieval, heterogeneous retrieval, and so on. Also, we can distinguish between approaches of document retrieval, that seek to return the entire document, and approaches of fragment/focused retrieval, that seek to return relevant information units instead of returning the entire document. These approaches are not exclusive; we can also mention others such as database retrieval, knowledge graphs and so on.

- **Result aggregation:** previous sub-task results are used as input for this last sub-task. The purpose of result aggregation is to build the final result, which will be returned to the user by putting these results together in the best and coherent way. In the following, we mention and describe some generic ways of content aggregation:

  - *Sorting:* is the most used method of representation in the field of information retrieval. It is used to process the list of returned fragments and return a new sorted list with respect to some defined features/functions, *e.g.,* relevance function, location, time *etc.* This vision is limited since the aggregated search aims to go beyond the ranking.
  - *Grouping:* based on common features of the returned results, the grouping action tends to create groups of results that share at least some

features or similar content, such as an event in the same time, same information format, same kind of object, same location, *etc*.

- *Merging:* we can present this process as the action that brings several fragments/results into one aggregate result. This last can be a new document, a summary off all relevant results or any other representation/format allowing the merging of results.
- *Splitting:* unlike the merging process, the splitting process decomposes fragments/results into other smaller ones. Typically, this process is used when the returned fragments are relatively huge and is usually augmented by another process, like the sorting or the grouping, that better represents the final results. The output of this process can be an ordered list of even more smaller fragments.
- *Extracting:* The main purpose of this process is to extract and identify one or more semantic information from the returned fragments. These semantic information can be entities describing places, pronouns identifying interlocutors, images or video, and so on.

4.2 Aggregated search: related research and IR disciplines

Aggregated search can be seen as a mature subfield of previous works in the Information Retrieval fields such as federated search, metasearch, semantic search, natural language generation, and so on. In the following, we discuss these different disciplines from the perspective of the aggregated search.

4.2.1 Federated Search

Federated search or distributed information retrieval is a multiple distributed data sources paradigm used in the IR field [167–169]. One can identify three tasks, as for the aggregated search, in this paradigm. Each submitted query undergoes the dispatching task that aims to select some sources, which seem relevant to the query, to search. These sources are indexed and characterized by a local representation that is used to identify them better. After that, the system sends the query to these selected sources, for which it may have specific matching algorithms, which return results for the query to the system. Finally, the obtained results are presented to the user. Typically, the final answer is a ranked list.

Using the concept of federated search in the web gives birth to two paradigms: meta-search [170] and aggregated search [171]. A metasearch engine queries several search engines, as an intermediary to query different sources, and combines results from them or display them separately [20]. Unlike aggregated search approaches, federated search approaches use assumptions that are not worth in the aggregated search environment. However, they are similar in the way of querying distributed sources and assembling the final returned results.

4.2.2 Cross-Vertical Aggregated search

At the onset of the Cross-vertical aggregated search ($cvAS$), studies have seen it as an instance of both federated search [169] and meta-search [170]. However, since the introduction of the concept of aggregated search, most studies, such as [172–175], classified the $cvAS$ within the aggregated search direction. Cross-vertical Aggregated Search is defined as the task of searching and assembling information from different vertical search engines in response to a user query [21, 175, 176]. Usually, this paradigm is related to the web search where each vertical represents a specific collection, such as videos, images, news and so on [158, 177]. Using $cvAS$ provides more visibility to vertical search engines [176] where the final results are more than a list of fragments.

4.2.3 Natural Language Generation

From the perspectives of the expected results, Natural Language Generation (NLG) approaches and Aggregated Search approaches are similar since they both use information fragments across several documents/sources rather than a list of ranked documents. The purpose of the Natural Language Generation is to organize textual information using predefined ways to generate answers in an appropriate linguistic form [178]. Here we show the analyses of NLG given in [21]. Approaches using NLG paradigm care less about the query dispatching and fragments retrieving as the information given beforehand from databases or search engines [179], that is, some NLG approaches start from a known context of use which means that the need is considered implicit. Based on the information fragments, *NLG* aggregates these fragments using different prototypical ways of information organization known as discourse strategies. These strategies can be either static, using cause-effect relation or chronological form, or dynamic that depend on the information need and the information availability also. To this end, the dynamic strategies, also known as learned strategies, use learning models on examples databases or documents that serve as a training set. Thus, each learning strategy is specific to one known domain [180]. Interested readers are referred to [178] for an interesting study of *NLG*.

4.2.4 Question answering

Unlike traditional information retrieval engines, that return a ranked list of documents, Question Answering (*QA*) paradigm provides a set of multiple responses [181]. One can see this paradigm like a case study for aggregated search since the answers are produced through information extraction and assembling. Similar to the aggregated search, we also find the three main sub-tasks of the defined framework (Figure 4.2) in the QA paradigm. In the query dispatching sub-task, several approaches are used to understand the query, identify named entities and other helpful facts within the query, which are useful for selecting potential sources. The fragment retrieval sub-task in QA aims to extract information, in the form of text, and assemble answers. Although this sub-tasks does not amount to the importance of the other two sub-tasks, it is worth mentioning that it is usually a critical and error-prone process, which is usually not easy. Finally, as a third sub-task, QA engines return a list of potential answers which juxtaposed with supporting text passages extracted from the matching documents. Further information about QA could be found in this interesting study [165].

4.2.5 Composite retrieval

Composite retrieval paradigm is an interesting new search paradigm that seems similar to the aggregated search paradigm. Its main purpose is to build a coherent set of item bundles, associated with different aspects or sub-topics of the query, rather than return a list of ranked documents. This paradigm was first introduced in [182]. In this study, the composite retrieval aims to return the k-diverse bundles of complementary items. For example, planning a trip requires a query that expects as answer bundles associated with travel options such as transportation, hotels, restaurants, the weather, points of interest, and so on. Each returned bundle should be valid within the constraints of the query, such as the time of the trip and the maximum budget to spend. This problem is NP-hard as showing in [182]. Several other works, such as [183, 184], process composite retrieval within the aggregated search context.

4.3 Aggregation search in graphs

Recently, several approaches, such as [7, 36, 185, 186], have been proposed to deal with the problem of graph query processing. Their main purpose is to find the information/subgraph that seems relevant to the query graph. In this respect, the main challenge is how to ensure the efficiency of the graph comparison process in terms of search time and search space?

In a scenario, like previous ones of aggregated search, where the answer is scattered across several fragments/subgraphs, most of the existing approaches do not attempt to assemble subgraphs in a sensed way to provide the final answer for a given query. In view of this context, several works seem to have a similar intention as the problem of aggregated search. This concept differs from one context to another, where the keyword *graph aggregation* has a different meaning. Usually, most of the proposed approaches aim at performing a certain level of summarization or compressing using merging techniques based on shared common features [187–191].

In [187, 188] authors propose a model that allows merging XML data (trees) streams. It is based on a merge template that defines the result document's structure, and a set of functions to be applied to the XML data streams. Similarly, the work involved in [189] performs aggregation in two steps. First, all nodes that share some common features are grouped within the same super-node. Next, if all the nodes within two super-nodes are related then a relation/link between these super-nodes must be created. Authors in [192, 193] present an XML keyword search model based on possibilistic and bayesian networks respectively, which allows returning, in the same unit, the answer with its complementary elements. The study given in [191] introduces a set of formalizing rules, called graph aggregation, to perform summarization preferences within a group of users. The main idea of this study is to build a final graph, which represents users with their common preferences, from a set of oriented graphs, where each graph represents preferences of a single user. Similarly, aggregating and merging found a way in the ontology field. The study presented in [194] performs some level of aggregating and merging in order to summarize a global choice of a group using opinions, preferences, and judgments available.

4.4 Relational aggregated search

Relational Aggregated Search (RAS) groups the approaches that involve retrieving and aggregating information fragments using their relations [173]. Combining both paradigms, entity-oriented search [195] and relational search [196, 197], gives rise to an interesting alternative that seeks not only entities of information, but also the relationships between these entities, which makes this new paradigm more efficient. In this respect, the relation between RAS paradigm and the graph search (graphs, knowledge graphs, graph databases *etc.*) cannot be overlooked by the fact that both seek information surrounded by structural requirements.

In the following, we present the main concepts used in RAS as well as an according framework of relational aggregated search.

4.4.1 Relational aggregated search framework

Relational aggregated search paradigm may improve many search directions, such as semantic search, entity-oriented search, database information retrieval, ... , by considering the relation between information fragments. The use of these relations can help, in most scenarios, to find more structured results by considering more attributes related to the query, *e.g.,* a query that seeks a person may have, as a result, not only a name but also any other type of information that seems useful and relevant to the query such as date and place of birth/death, news and so on.

By analyzing the needs and the requirements of this paradigm, we find that the aggregated search framework, depicted in Figure 4.2, can be generalized for the relational aggregated search paradigm as long as it takes into account the relation aspect within the three sub-tasks (query dispatching, fragment retrieval, and result aggregation).

4.4.1.1 Query Dispatching

The query dispatching sub-task maintains the same principle and the same main goal as described in Section 4.1. It aims to interpret and reformulate the used query before processing the query-sources matching to select sources deemed relevant. Inspired by the work in [196], authors in [173] propose a taxonomy that identifies three types of queries where benefits from relational aggregated search are obvious. In the following, we describe these relational query types.

- Attribute query: This type of query looks for units of information that can directly meet the desired need, for example "Game Of Thrones filming locations".
- Instance query: This type of query looks for a known instance of a class/ category. The returned result should contain this instance with all its related attributes, for example "Game Of Thrones" is an instance of the class Tv Series.
- Class query: This type of query is a generalization of the previous type. A class query could be as "French writer", "movies", "animals" and so on. Using this kind of query requires more voluminous answers, *i.e.,* the returned answers contain all class instances with their attributes.

The query dispatching is an important task since the query type determines which search approach should be triggered and which solution of result aggregated should be investigated.

4.4.1.2 Relation search

The second sub-task in relational aggregated search is called relational retrieval/ search (instead of fragment retrieval). In this sub-task, RAS uses approaches that focus on the relation between the information units. For this reason, interrogated sources are usually structured as entities with specific inter-relations. This kind of sources is known as knowledge bases. Another interesting information search field that uses relation-based search is known as Information Extraction (*IE*) [198]. *IE* is common to extract and relate information from structured sources. Existing approaches can extract not only information entities, such as names, location, *etc.*, but also their relations such as "George R. R. Martin" author of "A Song of Ice and Fire". However, further information about this search field could be founded in these interesting studies [21, 199].

4.4.1.3 Result Aggregation

The use of relations allows considering new ways to ensure the result aggregation. In *RAS*, the aggregation of results depends mainly on the type of the query. In the following, we discuss them briefly.

- *Result aggregation of the attribute query*: The best choice in answering to an attribute query is to return the exact answer, *i.e.,* the correct value of the attribute being requested. For this type of query, choosing the most relevant answer is a delicate decision. That is why returning a list of candidate answers can be considered the best choice.
- *Result aggregation of the instance query*: When the query is an instance, the answer can be the values of all the attributes of the instance. In such a case, the best choice can be the summarization of these attributes.
- *Result aggregation of the class query*: As described above, queries of type class could be answered by a list of instances that describe this class. This list could be presented in tables with rows representing the instances and columns representing their attributes.

4.5 Chapter summary

In this chapter, we presented the aggregated search starting from motivation and a general definition in its original context, *i.e.,* Information Retrieval. We have shown that the use of the aggregated search provides a rewarding experience for users. We presented and discussed a general framework where the aggregated search process is decomposed into three main sub-tasks (query dispatching, fragment retrieval, and result aggregation). Further, we presented some important and related works that go beyond information retrieval (query matching) with an additional effort on result aggregation. We have started with aggregated search in the IR field, such as federated search, and we have presented and analyzed a spectrum of approaches, from multiple research fields, in the context of aggregated search. Among these approaches, we can briefly mention federated search, cross-vertical aggregated search, question answering, natural language generation, composite retrieval and so on.

Furthermore, we introduced and discussed the concept of graph aggregation from the point of view of literature work. The interpretation of "aggregation" in these works is different from that in the information retrieval field. However, in the graph context, the aggregation can be achieved by graph operators, such as graph compression or graph summarization, which aim to merge entities (nodes) and relations (edges) that share some features.

At the end of this chapter, we presented the relational aggregated search, which is related to the graph search given that both look for information surrounded by relations. We highlighted and focused on RAS as an intersection between the graph search and the aggregated search, which represents the scope of our next contribution.

Chapter 5

Tree Matching and XML Retrieval

Contents

5.1 Introduction

Nowadays, the *XML* (eXtensible Markup Language) is among the most used format for representing and exchanging information given its simple nature and self-description. The advent of applications related to the emergence of the internet, ranging from intelligence web searching to e-commerce has brought this standard to greater prominence. The strength of XML lies in its ability to describe any data domain through its extensibility. XML tags describe the hierarchical structure of the content. An XML element is delimited by an opening and a closing tag, and it describes a semantic unit or a hierarchy of the document.

Victim of its success and intensive use, XML has shown the need for appropriate retrieval methods that can exploit huge documents. Using this standard, structured documents focus on relevant information. They contain heterogeneous contents organized with structural information. The structure of a document can be used to process textual information with more granularity than the entire document.

In this context, many approaches have been proposed to deal with XML retrieval. However, most of them have not exploited the benefits of this standard, although the use of graph theory may be of interest. In this respect, the XML data model allows considering documents as trees [200], which represent a particular kind of graph. In such a representation, nodes represent XML elements (*i.e.,* text) and edges represent relations between these elements. The same representation can be used to formulate the queries. On top of that, the retrieval process can be ensured by a matching process between the tree query and the tree documents.

In [18], the authors highlight three main challenges to deal with structured queries and the tree matching problem. The first issue is how to ensure efficiency in terms of returned results, especially when the interrogated documents are enormous. The second issue is how to interpret structural constraints when the structure of the query does not match the structure of the document; nevertheless, elements describing the information match even approximately the need in the query. At last, how to combine the content requirements of the queries, if they exist, with the structural one.

5.2 Querying XML data

XML documents can be considered as trees, which allows applying algorithms proposed for tree matching. This section is devoted to this topic. In the following, we will first recall some backgrounds and present some issues behind XML retrieval.

```
<article year = ‘1996”>
        <header>
                <title> An Algorithm for Subgraph Isomorphism</title>
                <author> J. R. ULLMANN </author>
        </header>
        <body>
                <section>
                        <title> Abstract </title>
                        <paragraph> Subgraph isomorphism can be determined by … </paragraph>
                </section>
                <section>
                        <title> Introduction</title>
                        <paragraph> Corneil and Gotlieb [4] mention that … </paragraph>
                        <paragraph> Simple Enumerating Algorithm for … </paragraph>
                        <paragraph> … </paragraph>

                </section>
                 <section>
                        …
                 </section>
                <section>
                        …
                </section>
        </body>
</article>
```

FIGURE 5.1: An example of XML document

5.2.1 Backgrounds on XML documents

As mentioned earlier, the data model of XML documents gives them a tree representation. In such a representation, the *root node* represents the whole document, *internal nodes* (*i.e.,* non terminal nodes) represent elements and *leaf nodes* (*i.e.,* terminal nodes) represent the contents. These nodes are linked with edges that represent their hierarchical relations. For more visibility, we give an XML document in Figure 5.1 and its corresponding tree representation in Figure 5.2. By the way, XML documents can be categorized into two main classes. The first class, called "data-oriented documents", is characterized by a determined structure and homogeneous contents. Documents within this category can be considered as a database where each element represents a database record (i.e., like a couple of key-value). Unlike the first class, documents in the second class, called "text-oriented documents", have irregular structure and may contain mixed contents.

Generally, the success of an information search application depends directly on the data representation and the efficiency of the search process. Talking about XML retrieval leads to two most problems in XML access approaches: exact and inexact matching according to their results. In other words, exact approaches return results that satisfy all the requirements of the query. Such approaches focus on efficiency problems, and they are more concerned with searching than ranking

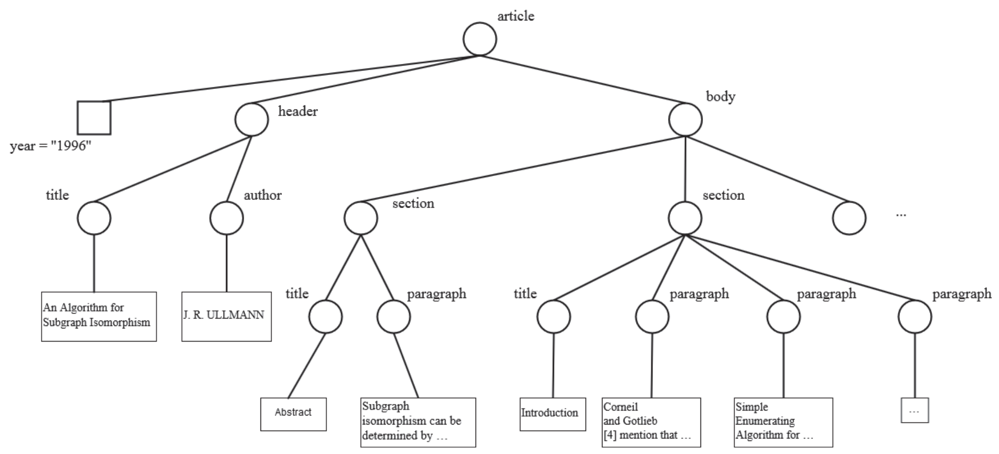

article
year = "1996"
header
body
title
author
section
section
...
An Algorithm for Subgraph Isomorphism
J. R. ULLMANN
title
paragraph
title
paragraph
paragraph
paragraph
Abstract
Subgraph isomorphism can be determined by ...
Introduction
Corneil and Gotlieb [4] mention that ...
Simple Enumerating Algorithm for ...
...

FIGURE 5.2: XML tree associated with the document of Figure 5.1

(*e.g., database search*). However, inexact approaches aim at ranking components, that deemed similar to the query, according to their relevance (*e.g.,* information retrieval approaches).

The XML retrieval problem is to match a document tree with a query tree. In the following, we will first present some query languages, and then we will describe some approaches from the information retrieval and the database communities.

5.2.2 Query languages

The user's need can be formulated through one of two types of queries, content-only or content-and-structure queries. Usually, content-only queries are used in traditional information retrieval, where simple keywords terms express the desired information. Such queries are suitable for XML retrieval, especially when users do not look or know the structure. In our work, we are not interested in this kind of queries, since they do not have structural hints. In content and structure queries, the content conditions are surrounded by structure requirements. These reduce the size of returned results by favoring those that respect, in addition to the content, the structural constraints. In [201], the authors classified the content-and-structure query languages in three main categories:

- *tag-based queries* is used to express simple conditions concerning the tag of the information sought, *i.e.,* "title: game of thrones" which means that the user looks for a title element about "game of thrones".

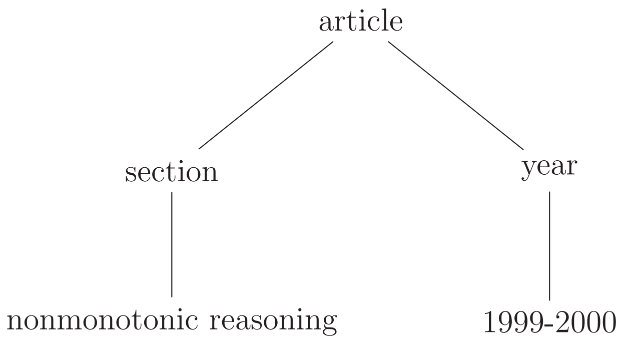


FIGURE 5.3: INEX topic expressed as an XML fragment.

- *path-based queries* use the XPath syntax [202] to describe the contents. Among the query languages that use this syntax, we can cite FuzzyXpath [203] and NEXI language [204].
- *clause-based queries* express the user needs via clauses that have structure similar to the SQL one. We can mention, as examples, XQuery [205] and XQuery full-text [206].

Authors in [166] argue that whatever the query language used, content-and-structure queries, can be represented as XML fragments, in other words, as labeled trees. Using this representation of queries offers more flexibility in terms of representation and approximation of information needs. For instance, consider the following need from the INEX topic: "Retrieve all articles from the years 1999-2000 that deal with works on nonmonotonic reasoning.". This query can be translated into the fragment shown in Figure 5.3.

The retrieval process can be considered as a tree matching process. In the next section, we describe some works from the literature on exact and approximate tree matching.

5.3 Algorithms for tree matching

The tree matching problem is defined as the process of finding a mapping between the nodes and the edges of two trees that satisfy some (more or less stringent) constraints, ensuring that substructures in the one tree correspond to similar substructures in the other tree. This problem is crucial in many applications where the information are represented and structured by trees. In order to state the problem, the terminology of trees and their components has to be defined first.

Definition 7. In graph theory, a *tree* $T = (V, E)$ is a connected graph without cycles where any two nodes are connected by exactly one path. If you suspend a tree by a node $r \in V(T)$, you obtain a *rooted tree* on r. In a rooted tree, a hierarchical relationship exists between any node and its neighbors. Then each node x (except the root r) has exactly one *parent* in the tree (*i.e.* its predecessor in the path from r to x), denoted $parent(x)$, and the remaining of its neighbors are its *children*. If a node $x \neq r$ has no child, it is called a *leaf*. Thus *sibling* nodes x, y satisfy $parent(x) = parent(y)$ and an *ancestor* of a node x is a node in the path from r to x (except x). Note that an *independent ancestor* of v is an ancestor that conveys only the v's information. Moreover a tree is said *labeled* if each node is given a label. A tree is said *ordered* if the order between siblings is important, otherwise the tree is *unordered*.

Tree matching approaches are classified into two main categories. The first one represents exact tree matching, which requires a strict and exact correspondence among the two trees, or their sub-trees, to be compared. The second one concerns inexact tree matching or approximate tree matching. In this category, two sub-trees can be matched even if they are structurally different to some extent. In the following, we present the tree matching problem, which is one of the most interesting problems in graph theory. We also survey and discuss the exact and approximate tree matching algorithms proposed to solve this problem.

5.3.1 Exact tree matching

Exact tree matching approaches aim to find out if a mapping is edge preserving for each mapping for each node of the two trees. In other words, it requires a strict correspondence between the two trees being matched, or at least between sub-trees of them.

We give below the formal definition of the exact tree matching problem.

Definition 8. Let target $T = (V, E)$ and pattern $Q = (V_q, E_q)$ be two ordered labeled trees. The pattern tree Q matches the target tree T at node r if there exists a one-to-one injective function from the nodes of Q into the nodes of T such that:

- the root of Q maps to r,

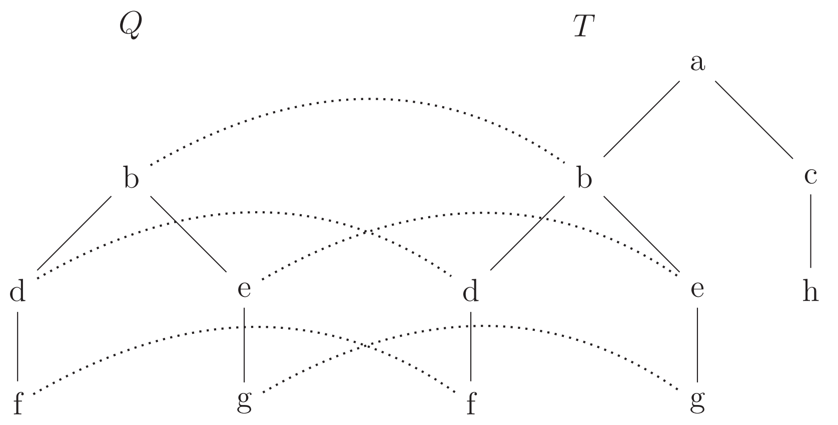


FIGURE 5.4: An example of exact tree pattern matching.

- if $u \in V_q$ maps to $v \in V$, then u and v have the same label,
- if $u \in V_q$ maps to $v \in V$ and v is not a leaf node, then each child of u maps to some children of v.

A fraction of a target tree is shown as T in Figure 5.4, and the requirement of the pattern tree is shown as Q in Figure 5.4. Dotted lines represent the one-to-one mappings between T and Q.

Tree matching process takes, in the worst case, $O(nm)$ time where n is the size of the target tree and m is the size of the pattern tree. This process is based on a naive idea that consists of visiting all the nodes in a pre-order walk. For each visited node v, the algorithm seeks recursively for a possible occurrence of the pattern at the node v and the process will be terminated if a mismatched is detected.

Several works have been done to improve the naive tree matching algorithm. These works fall into one of these two categories: *traversal approaches* or *decomposition approaches*. For example, but not limited, we mention some approaches. In the traversal approaches, authors proposed in [207] *bottom-up* and *top-down* matching algorithms. The *top-down* algorithm encodes all the root-to-leaf paths as strings and seeks for occurrences of the pattern string in the target-tree string using a string pattern matching algorithm. The *bottom-up* algorithm aims to find, for each node in the target tree, all patterns and all parts of patterns that match this node. However, the main idea of the decomposition approaches is divide and rule. Approaches that follow this mechanism try to decompose the target tree in small pieces in order to make the matching process easier. Authors proposed in [208] a two-phase algorithm for tree pattern matching. The first step aims to construct a matching automaton from a given pattern set. In this respect, it generalizes the string matching algorithm of [209] using the position ordering relation for the decomposition of patterns. After that, these patterns are merged into an automaton. Next, in the second step, target trees are fed into the automaton, by traversing through the automaton according to the target trees.

We will not focus here on the exact tree matching problem. However, we direct the interested reader to [18] and references therein for additional information about this research field. In the following, we review approximate tree matching algorithms.

5.3.2 Approximate tree matching algorithms

Approximate tree matching, or inexact tree matching, can be defined as the process of finding the most similar matches of one tree/subtree against another one. This process can be interpreted as the task of measuring the dissimilarity between trees. One of the most used metrics for approximate tree matching is the tree edit distance. In [210], the authors generalized the edit distance between strings used in [211] for trees. The edit distance between two trees is defined as the minimal cost of edit operations that transform a tree into another tree. There are three basic edit operations: node insertion, node deletion and node relabeling. We describe each of these operations in detail below.

Let T, q be two labeled trees. The tree edit distance between T and q is defined as:

$$\delta(T, q) = \min_{e_1,\ldots,e_n \in \gamma(T,q)} \sum_{i=1}^{n} cst(e_i),$$

where $\gamma(T, q)$ is the set of edit operations, and $cst()$ represents the edit operation cost. The edit operations are:

- Insertion: insert a node v as a child of w in T, w's children become v's children.
- Deletion: delete a node v in T, v's children become children of v's parent.
- Relabeling: replace the label of a node by another label.

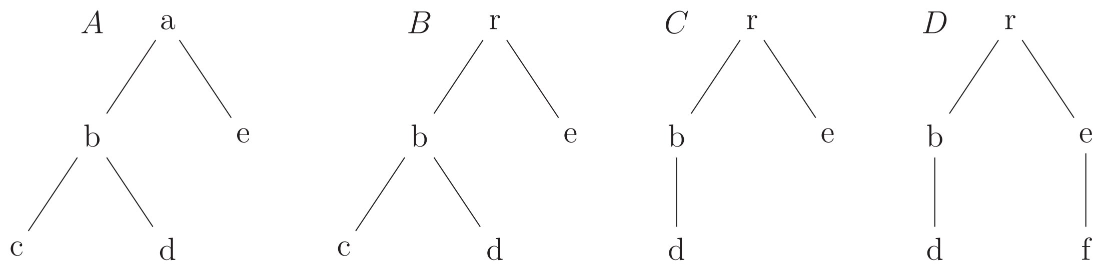


Figure 5.5: Tree edit distance operations

Figure 5.5 represents an example of the three edit operations, where the transition $(A) \longrightarrow (B)$ represents the relabeling of the label a by r, $(B) \longrightarrow (C)$ represents the deletion of the node labeled by c and $(C) \longrightarrow (D)$ shows the insertion of node labeled by f.

Approximate approaches of tree matching can be classified, based on the concept of editing distance, into three classes [212]: tree edit distance approaches, tree inclusion approaches, and tree alignment distance approaches.

Tree edit distance approaches. As mentioned earlier, the tree edit distance, $\delta(F, G)$, is defined as the minimum-cost sequence of node edit operations that turns F into G. The recursive approach for this problem was first proposed in [210], and the solution takes $O(|F| \cdot d_F^2 \cdot |G| \cdot d_G^2)$ time and space. Authors had improved in [213] the previous algorithm and the resulting algorithm has a complexity of $O(|F| \cdot |G| \cdot \min\{l_F, d_F\} \cdot \min\{l_G, d_G\})$ time and $O(|F| \cdot |G|)$ space. In this approach, the distance between two trees F and G is computed from the solutions of following smaller sub-problems: *(1)* $\delta(F - v, G)$, *(2)* $\delta(F, G - w)$, *(3)* $\delta(F_v, G_w)$, *(4)* $\delta(F - F_v, G - G_w)$, and *(5)* $\delta(F - v, G - w)$. Figure 5.6 is an illustration of the this recursive solution.

Given the importance of the problem and its presence in many emerging applications, researchers are constantly coming up with remarkable improvements. Author in [214] proposed a faster algorithm, for un-rooted ordered trees, with $O(|F|^2 \cdot |G| \cdot log|G|)$ time complexity and $O(|F| \cdot |G|)$ space complexity.

The dynamic programming implementations of the recursive solution show their effectiveness as the fastest algorithms. Given that each sub-problem is computed, from other sub-problems, in constant time, the complexity of these algorithms is equal to the number of different relevant sub-problems they produce [215]. Authors showed in [216] that algorithms proposed in [213, 214] could be described

$$
\begin{aligned}
&\delta(\emptyset,\emptyset) = 0,\\
&\delta(F,\emptyset) = \delta(F-v,\emptyset) + c_d(v),\\
&\delta(\emptyset,G) = \delta(\emptyset,G-w) + c_i(w),\\
&\textit{if } F \text{ is not a tree or } G \text{ is not a tree:}\\
&\delta(F,G) = \min \begin{cases} \delta(F-v,G) + c_d(v) & (1)\\ \delta(F,G-w) + c_i(w) & (2)\\ \delta(F_v,G_w) + \delta(F-F_v,G-G_w) & (3),(4) \end{cases}\\
&\textit{if } F \text{ is a tree and } G \text{ is a tree:}\\
&\delta(F,G) = \min \begin{cases} \delta(F-v,G) + c_d(v) & (1)\\ \delta(F,G-w) + c_i(w) & (2)\\ \delta(F-v,G-w) + c_r(v,w) & (5) \end{cases}
\end{aligned}
$$

FIGURE 5.6: Recursive formula for Tree Edit Distance.

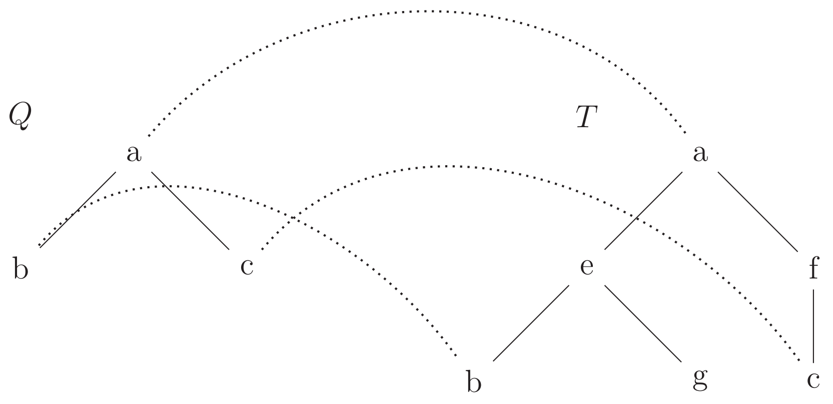


FIGURE 5.7: An example of tree inclusion matching.

as decomposition strategies. They proposed a novel algorithm that minimizes the number of distinct recursive calls and runs at most in $O(|F| \cdot |G| \cdot log|G| \cdot log|G|)$ time in the worst case, and $(|F||G|)$ on average.

Tree inclusion. The tree inclusion is a particular case of the tree edit distance. Let F and G be two labeled rooted trees. The tree inclusion aims to locate subtrees of G that are instances of F. This problem can be reduced on tree edit distance since one can only use sequences of delete operations to obtain F from G. Depending on the order between siblings of the pattern nodes, this problem has two versions: the ordered tree inclusion problem, where the order is important, and the unordered tree inclusion, where the order is not important. The study in [217] focused on these two versions and showed that the problem for unordered trees is NP-complete. Figure 5.7 shows an example of tree inclusion.

Let F and G be two labeled rooted trees. The inclusion of F in G is defined as the

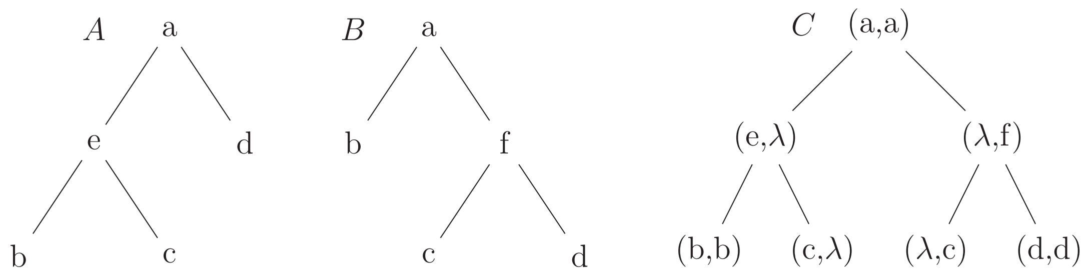


Figure 5.8: Tree Alignment Distance: (A) tree F, (B) tree G, (C) an alignment of F and G.

embedding relation of F into G, *i.e.,* there is an injective function $h : F \longmapsto G$ that preserves labels and ancestor-ship. For each $u \in F$ and $v \in F$, the following requirements are preserved:

- $h(u) = h(v)$ if and only if $u = v$,
- the labels of the nodes u and $f(u)$ are identical,
- v has an ancestor u in F if and only if $f(v)$ has an ancestor $f(u)$ in G

Tree alignment distance. The tree alignment problem is another particular case of the tree edit distance problem. The principal requirement of this concept is that all sequences of insertion operations must be done before any delete operation. An alignment of two labeled trees, F and G, consists first in making them isomorphic by the insertion of nodes, on both of them, labeled with the null label λ, and then overlaying the first augmented tree on the second. The cost of the alignment can be defined as the sum of costs of all opposing labels in the alignment and the purpose in such a problem is to minimize the final cost. Figure 5.8 illustrates an example of an ordered alignment [218].

We encourage readers interested in more details to consult [18, 212] for surveys as well as references therein.

5.4 Tree matching for XML retrieval

As a data structure, XML is increasingly used to model data and complex objects. Tree representation of XML documents allows conveying as much information as possible to ensure an efficient representation of complex objects and also a relevant comparison between objects. Thus various real applications use XML as a model of representation. In many cases, the success of an application based on a tree representation of data is mainly dependent on the efficiency of the underlying tree query processing. Talking about tree query matching leads directly to one of the most popular problems in graph theory, which is the tree matching problem. Tree matching consists of finding the correspondences between the nodes of two trees that provide the best alignment of their structures. Generally, tree matching methods can be divided into two main categories. The first category covers the exact algorithms that consist, in the field of XML retrieval, of finding all twig patterns in XML document. The second category covers the inexact algorithms that return a ranked list of the most similar matches.

In this section, we present and discuss state of the art related to the tree matching algorithms in the field of XML retrieval. Since this part of this thesis is situated initially in the graph theory and aggregated search fields, we highlight and focus on inexact tree matching algorithms as a preliminary step for the aggregated search in XML documents.

5.4.1 Exact tree-matching algorithms for XML retrieval

As we have seen previously, the underlying data model of XML is a labeled tree, and twig patterns are used for expressing queries. Recently, the problem of finding all occurrences of such a twig pattern in an XML database has been a great deal of interest. According to the study given in [219], most of the proposed approaches can be classified into four groups:

- structural join approaches,
- holistic twig join approaches,
- sequence matching approaches,
- other important exact tree algorithms.

In the following, we cite some of the well-known algorithms and discuss their results.

5.4.1.1 Structural join approaches

Structural join approaches are based on a fundamental idea, which usually requires three steps: (1) decomposition step, (2) matching step and (3) merging step. In the decomposition step, a twig pattern is decomposed into a set of basic parent-child and ancestor-descendant relationships between pairs of nodes. Then, in the matching step, each relationship is separately executed using structural join techniques and the results of this process are stored for further processing. In the final step, the intermediate results are merged in order to produce the final result.

Multi-predicate merge join (MPMGJN) [220] is the first structural join algorithm that aims to find all occurrences of the basic structural relationships. This algorithm takes advantage of the containment labeling scheme [220] (called region encoding) that encodes each element in an XML database by its positional information. The main purpose of this labeling scheme is that the structural relationship (ancestor–descendant and parent–child relationships) between two elements can be determined easily without knowledge of the path information between these two elements. It uses a region code (start, end, level) to represent the position of an XML element in the data tree where start and end are generated by performing a pre-order traversal procedure of the data tree, and level is the nesting depth of the element in the data tree. The result of this algorithm showed that for many XML queries, MPMGJN is more than an order-of-magnitude faster than the standard Relational Database Management System join implementation.

Using the same containment labeling scheme of XML elements, authors developed in [221] two algorithms, Tree-Merge and Tree-Stack, for matching parent-child and ancestor-descendant structural relationships, which decrease the time of join processing.

5.4.1.2 Holistic twig join approaches

The main problem of approaches based on decomposing twig queries into multiple binary relationships is that they produce large and possible unnecessary intermediate query results even when the input and output size are more manageable. In order to avoid this problem, authors in [222] proposed the first holistic XML twig pattern matching algorithm, wherein, no large intermediate results are generated. The main purpose of this approach is avoiding storing intermediate results unless they contribute to the final results. To this end, authors in [222] use a chain of linked stacks to compactly represent intermediate results of individual query root-to-leaf paths, which are then composed to produce the final results. The experimental results have shown that TwigStack is more efficient than all sequential algorithms that need the entire tree.

Several works, such as [223–230], have adopted the idea of holistic twig join in order to provide more efficient algorithms.

5.4.1.3 Sequence matching approaches

Using an indexing method, the sequence matching approaches transform both queries and XML documents into sequences, and the matching process is reduced to subsequence matching problem. In [231, 232], the authors use the pre-order and post-order ranks to transform the tree structures and use the sequence inclusion algorithms for strings. The whole process consists of three main steps. The first step is to decompose the query into a set of root-to-leaf paths so that the tree inclusion can be safely applied. The second step is to evaluate the set of paths against the data signature. The third step is to generate the final results by joining compatible intermediate solutions. Experiments in [231] demonstrate the efficiency of the decomposition approach.

Several improvements, such as [233–235], have been proposed to avoid expensive join operations using the tree structures as the basic unit of the query.

5.4.1.4 Other important exact tree algorithms

Several other works have been proposed to handle the tree matching over XML documents. Authors proposed in [236] a novel method, called S^3, which can selectively process the document nodes. In this approach and unlike all previous methods, path expressions are not directly executed on the XML document, but they are first evaluated against a guidance structure, called QueryGuide. The generated information represent an abstraction of the XML document that describes the structure of the document by its paths, *e.g.,* the nodes of XML data are labeled by Dewey labeling scheme. The experimental results of S^3 and its optimized version outperform previous tree pattern processing method in term of response time, I/O overhead and memory consumption-critical parameters.

In [237], the authors propose a version-labeling scheme and TwigVersion algorithm to deal with the problem of processing huge documents. The main idea is to encode all repetitive structures in XML documents, which allows avoiding a large number of unnecessary computations. Both theoretical proof and experimental results reported in this study demonstrate the effectiveness of the proposed method in terms of response time and memory consumption.

With the advent of the XML standard, XML query processing has attracted the interest of many researchers, and a rich literature has arisen throughout the years. In this section, we presented and discussed some well-known tree pattern processing algorithms. As mentioned before, an enormous volume of algorithms was proposed, and it seems nearly impossible to consider all related works. We encourage readers interested in more details to consult [18, 238–240] as well as references therein.

5.4.2 Approximate tree matching algorithms for XML retrieval

The stringent conditions imposed in the exact tree matching are too strict to find occurrences of queries in the target documents. Looking at data sources nowadays, like XML documents for example, one can argue that these sources can undergo brutal changes or deformations due to several reasons, such as noises in the data acquisition process, incomplete or missing information and so on. In such a situation, the matching process should be able to identify relevant answers, despite the existence of some differences between the query and its ideal answers, by relaxing the constraints imposed by exact matching algorithms.

Approximate matching algorithms may consider correspondences that do not satisfy, to some extent, the edge-preservation and/or label-preservation. In such a scenario, results are evaluated by a cost function, and the used algorithm must find a mapping that minimizes the matching cost.

In the literature, one can distinguish two principal classes of XML retrieval. Approaches of the first class can be considered as adaptations of traditional information retrieval to structured retrieval. Looking closely at these approaches, most of them use the tree representation of documents and queries but do not use algorithms for tree matching [18]. For example, the XFIRM system presented in [241] is based on a relevance propagation method for both keywords and keywords-and-structural constraints queries. In XFIRM, the scores are propagated from the leaf nodes to the inner nodes. We encourage readers interested in this kind of approaches to consult [242–246] as well as references therein.

Approaches of the second class make explicit use of graph matching algorithms. Most of them provide a two-step matching search process. The first step is the preprocessing of the query and/or the document. The second step aims to identify potential matches. In addition to reducing the time and space complexity of existing algorithms, preprocessing allows reaching more results by widening the search space, which is ensured by relaxing content and/or structural constraints. When considering structure, one can identify three types of relaxation: (1) order relaxation that consists in ignoring the order constraints between sibling nodes, (2) nodes relaxation that regroups the delete and the relabeling nodes operations and (3) edges relaxation, which is equivalent to add edges between nodes.

In the following, we list and review some leading and well-known approaches of XML retrieval that make explicit use of graph matching algorithms.

5.4.2.1 Approaches based on graph matching algorithms

In order to answer queries against XML documents, authors present in [247] a structural-pattern matching language that returns approximate answers. In this study, the matching process is seen as an unordered tree inclusion problem that uses a cost model to rank matches that seems relevant according to their similarity to the query. Contrary to the tree inclusion problem, which is not well suited to the XML retrieval, the tree edit distance is the most adapted for approximate XML retrieval [18]. As we mentioned above, the tree edit distance is one of the most used metrics for approximate tree matching. This problem [210] is an extension of the edit distance between string defined in [211].

In [107], the authors introduced the *XFinder* system which aims to find the top-k approximate matches of small queries in large documents. This system uses a revised *Prufer sequences* [109] to transform the query and document to strings and the tree edit distance is approximated by the longest subsequence distance between the resulting strings.

As against to this adaptation and based on *Levenshtein* editing distance [211], the authors introduce in [248] a weighted editing distance and reduce the XML-path matching process to a string edit distance scoring process in order to ensure approximate structure matching. Conjointly, the content matching is ensured by a strict and fuzzy searching based on the IDF of the researched terms. Then, the final score is obtained using a weighted linear aggregation function that combines the two previous scores.

Answering top-k queries is an active research area. The purpose of studies like [99, 249–251] is to find answers (subtrees) in a document (or collection) that seem relevant to the query. In [99, 249, 250], queries are in the form of keywords while the answers are subtrees that contain at least one used keyword. In such a case, two or more keywords may appear in different subtrees of the XML tree, and the results are ranked based on a content score and a structure score. Authors study in [249] the threshold algorithms [90] to combine the content and structure ranking. Authors defined in [251] a model that combines a structural score, using the tree edit distance and a classical content similarity, obtained by the propagation of the scores of the leaf nodes to their ancestors, in order to evaluate the similarity between queries and documents.

In search of answering approximate queries over XML data, authors in [252, 253] applied the tree edit distance conjointly with a certain level of relaxation in order to query a set of heterogeneous data sources with incomplete or missing data. In these studies, authors defined the Minimum Spanning Trees [254] as the tree that contains query elements and a minimum number of nodes/arcs. This adaptation requires a nodes scoring step and tree matching process. The node scoring depends on the type of nodes. In the case of non-leaf nodes, authors use Levenshtein distance [211] between query and document node name, and in the case of leaf nodes, authors use the fuzzy set theory to evaluate the membership degrees of nodes to the content conditions of the query. Then, a structural validation of the result is ensured by keeping nodes that build a Minimum Spanning Tree.

In [255], the authors propose a query relaxation method incorporating structures and contents over XML data. In this study, the similarity relation assessment has done by analyzing the inherent semantics. Instead of giving equal importance to each node to be relaxed, a relaxation ordering is maintained. In particular, the first relaxed structure to be considered is the one that has the highest similarity coefficient with the original query, and the first node to be relaxed is the least important. Authors designed a clue-based directed acyclic graph (CDAG) to generate and organize structure relaxation, and they defined a ranking factor to find out approximate and relevant queries.

ArHeX [256] addresses the problem of identifying the portions of documents that are similar to a given structural pattern. The pattern is mainly represented by a labeled tree where labels represent information that should be retrieved, and the relationships are imposed by the hierarchical structure. Firstly, pattern labels are used to identify the portions of documents in which the nodes of the pattern appear. Then, the extraction of subtrees (called fragments) is carried out by the exploitation of the ancestor/descendant relationships existing among nodes in the target which have labels similar to those in the pattern. The structural similarity between the pattern and the fragments is then evaluated for ranking the identified fragments and producing the results.

TASM [110], Top-k Approximate Subtree Matching, is one of the most effective approaches for identifying the k best approximate matches of query tree in document trees. For a given query tree, *TASM* returns the k subtrees of the documents tree (consisting nodes of the documents with their descendants) that are closest to the query tree according to canonical tree edit distance. Given the hypothesis, the most crucial issue is to be able to have a runtime cost linear in the size of the document and space complexity that does not depend on the document size.

5.5 Chapter summary

In this chapter, we tackled the problem of tree matching algorithms related to XML retrieval. We first presented important concepts and background concerning the querying of XML data. We then gave a structured overview of the most known algorithms for exact and approximate tree matching. We also presented and discussed the most important approaches for XML retrieval using tree matching.

In the next chapter, we will introduce and discuss a new approach to deal with the aggregated search in XML documents.

Chapter 6

A new Approximate XML Retrieval based on Aggregation Search

Contents

Among all available formats used to represent information, the XML format is now extensively used. The strength of XML lies in its ability to describe any data domain through its extensibility. Querying XML data sources proves to be an important task. Moreover, the noisy nature of those sources makes the approximate tree matching tools highly required in order to alleviate restrictive query answering. Several paradigms along with their appropriate algorithms have been proposed, but only a few of them try to aggregate a set of XML fragments in order to provide more significant and coherent answers. In this chapter, we introduce and propose a new framework for XML querying based on aggregated search. We start by giving the motivation behind the use of the aggregated search paradigm in querying XML data sources. We explain the benefits of using such a paradigm compared to traditional XML querying settings. Then, we present our proposed aggregated search XML querying framework. We first introduce and define the theoretical terms and concepts used in the framework. Then, we present the retrieval and aggregation processes and explain all the underlying routines and steps. After that, we analyze our proposed approach and conduct extensive experiments on real datasets, with different sizes, to attest the efficiency and quality of the proposed approach. Finally, in the last section, we give a chapter summary and outline several perspectives and future work directions.

6.1 Querying XML data using aggregated search

With the increasing popularity of XML for data representations, there is much interest in searching XML data. Usually, querying XML data sources can be easily achieved by using powerful database-style query languages, which consider the exact matching of documents and queries. However, searching XML data becomes intractable in practical applications since the hierarchical structure may be heterogeneous and data sources are usually noisy. Besides, any slight misunderstanding of the document's structure can easily lead to unsatisfactory queries and it can result an empty-set answer problem. Moreover, this problem requires to have complete knowledge of the data structure, which is not always the case. To deal with these restrictions, approximate matching approaches, which aim at ranking document components according to their relevance to the query, are widely used in a broad spectrum of real-world applications.

The vision of classical XML retrieval approaches is limited. Relevant information may not be contiguous in the same list of results, as it may be scattered across several data sources and/ or several documents. In such a case, the returned answer is just a group of elements that may not meet user expectations. By the way, few works on XML querying have addressed the XML retrieval (tree matching problem) by building an answer to a given query based on the aggregation of several results. The idea is to build, as far as possible, a more complete and richer answer by combining several results (subtrees). This search paradigm is known as aggregated search in XML retrieval (querying data in tree form) [257].

In the following, we first define the aggregated search in the context of querying data in tree form, like XML documents, then we give the key motivation behind the aggregated search paradigm in XML retrieval.

6.1.1 Definition

Aggregated search is a recent search paradigm where the information is not only retrieved but also assembled [21]. In other words, the goal of the aggregated search is to provide an integrated search across multiple data sources and build an aggregate response that contains shards of relevant answers and their complementary information to be returned in a unified document and a common presentation of results. Google[1] was the first who explicitly introduced the idea of aggregated search as universal search. Moreover, the first definition of the aggregated search was given in [20].

In the context of XML documents, the aggregated search problem seems to have similar intentions as that of the relational aggregated search. In a nutshell, aggregated search in XML documents refers to the process of retrieving information with more granularity and aggregate them to build the final result.

Typically, the process of querying XML documents uses the tree representation of documents and queries, whether implicitly or explicitly. With such a context and such a representation of data, Aggregated Tree Search can be defined as follows: Let q be a query tree, and F be a set of fragments (subtrees) that approximately match the query q. Aggregated tree search aims to build, from the set F, a result tree called the *aggregate* A that contains the most relevant elements and their complementary information so as to cover the expectations of the query. The aggregated tree search process will be further detailed and explained in the following sections, even so, we give an illustration of this process in Figure 6.1.

[1]http://googlepress.blogspot.com/2007/05/google-begins-move-to-universal-search_16.html

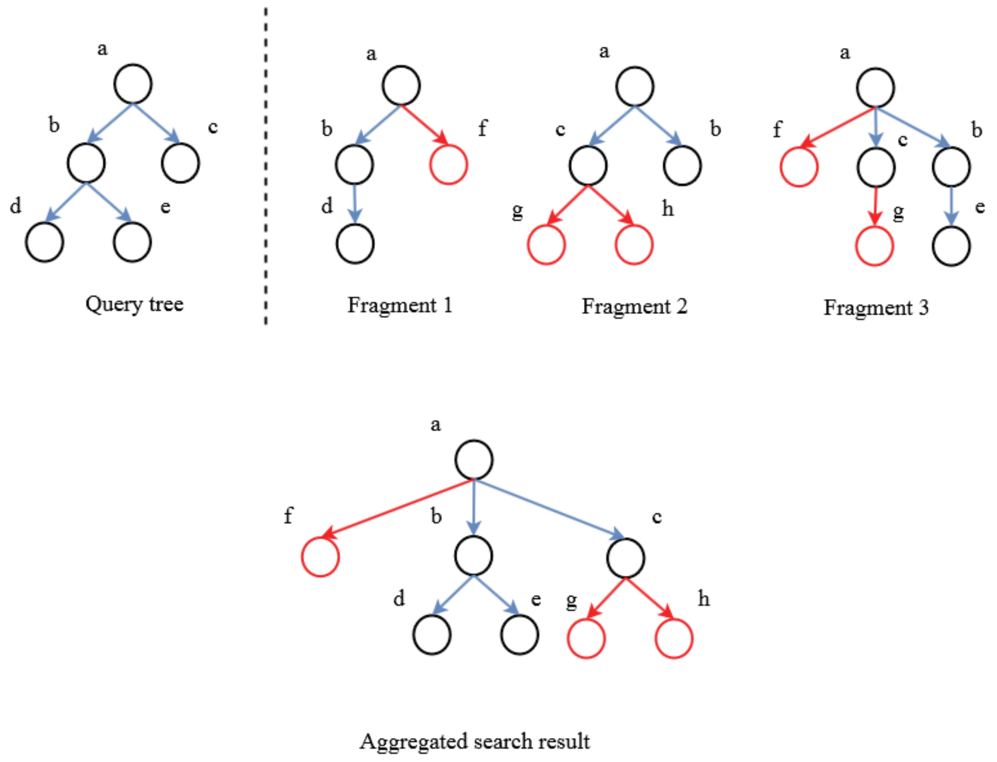


Figure 6.1: Aggregated tree search

From Figure 6.1, one can clearly see the difference between aggregated tree search and traditional tree search paradigms, *e.g.,* exact tree matching. Using aggregated search, one can still find matches with complementary information to a given query, whereas with traditional paradigms no result or only a part of results could be found since traditional tree search approaches are query wise, *i.e.,* each returned fragment is said to contain the query/part of the query or not, while in aggregated tree search, each fragment is to contain the query/ part of the query with complementary information, if any, to cover the expectations of the query.

6.1.2 Motivation

The strength of XML lies in its ability to describe any data domain through its extensibility and its appropriate tree representation. It becomes among the most used format for representing and exchanging information given its simple nature and self-description. Thus, methods for querying XML documents in order to explore knowledge within and make processing over these data are required. In this context, many approaches, varying from exact algorithms to heuristics, have been proposed. Usually, the success of an application based on a tree representation of data is mainly dependent on the context and the efficiency of the underlying tree query processing. Talking about querying data in tree form leads directly to one of the most widespread problems in graph theory, which is tree matching. Tree matching consists of finding the correspondences between the nodes of two trees, which provides the best alignment of their structures. Generally, tree matching methods can be divided into exact and inexact/ approximate algorithms. However, few works have tackled the problem of aggregated search in the XML context. One can see the problem of aggregated search in the context of XML documents as a tree matching task, in the way that it takes a content-and-structure query and try to find potential intermediate matches (called fragments), augmented by the process of building an aggregate, that contains the most relevant shards and their complementary information, as a final result.

We can summarize the motivation behind the aggregated search in the field of XML retrieval in two key points. First, using the aggregated search principle provides a significant benefit by producing more results that can match parts of queries when it is not obvious to find full matches. Secondly, with large-scale networks and massive data size, aggregated search becomes indispensable for discovering and exploiting information, especially when this information is distributed across several sites and scattered over multiple documents.

6.2 Preliminaries

In this section, we present some notions used in our approach. We first recall the description of an XML document. Then we give the formal definitions of the data structures used in the rest of the chapter: the query, the target trees, and the answer set.

In XML documents, tags describe the hierarchical structure of the content. An XML element is delimited by an opening and a closing tag, and it describes a semantic unit or a hierarchy of the document. An XML document can be represented as a labeled tree [17] (Definition 7). This last contains a root node that represents the entire document, internal nodes that organize the structure and leaf nodes describing the information conveyed by paths.

Query. The query q is a rooted labeled tree $q = (V_q, E_q, l_v)$, where (1) V_q and E_q are the sets of query nodes and edges, respectively, (2) l_v is a labeling function such that for each node $u \in V_q$, $l_v(u)$ is a label from the set of labels Σ.

Target trees. Target trees are represented by the set $L = \{f_{i_1}, ..., f_{i_j}\}$ of p trees (documents) which are referred as fragments f_{i_j}, $i \in [p]$ and f_{i_j} is a fragment from the tree p_i. Each fragment $f_{i_j} = (V_{i_j}, E_{i_j}, l_v)$ is a rooted labeled tree.

Aggregate. The expected answer is a tree (document) $A = (V_a, E_a, l_v)$ where $V_a \subseteq \bigcup_{i,j=1}^{i \leq p, j \leq k} V_{i_j}$ and $E_a \subseteq \bigcup_{i,j=1}^{i \leq p, j \leq k} E_{i_j}$ is a set of nodes and a set of edges from the target trees, respectively, and l_v is a labeling function such that for each node $u \in V_a$, $l_v(u)$ is a label from the finite set of labels Σ.

Postorder Queues. A postorder queue [110] is defined as a sequence of (label, size) pairs of the tree nodes in postorder. Let T be a tree of size n, the postorder queue, denoted $post(T)$, of the tree T is the sequence of pairs $((l_1, s_1), (l_2, s_2), ..., (l_n, s_n))$, where l_i is the label of the node t_i and s_i is the size of the subtree originated from t_i, with t_i being the $i-th$ node of T in postorder. The only operation allowed is *dequeue*, which removes and returns the first element of the sequence.

6.3 Approximate query processing based on aggregated search

In this section, we introduce and present our proposed approach to solve the problem of aggregated search in XML documents. Recall that our aim is to construct a final aggregate from the results of a query q sent to a collection $C = \{D_1, D_2, \ldots, D_p\}$ of documents. This process is performed in two main phases. The first one is the *retrieval process*. For each document $D_i \in C$, this phase gives a list L of target trees (each element from this list is called a *fragment*). The second phase is the *aggregation process*, where fragments from a list L are selected to be aggregated in the final document.

6.3.1 Retrieval process

Dealing with XML documents may highlight some challenges, especially with massive data. As data are growing in size, the number of matches can be excessively large. Inspecting all the results is a daunting task. Thus the users generally have an interest in the best-returned responses. Furthermore, the desired answer for a given query may be dispersed in the same document. These highlight the need to find, from each document, several fragments ranked by given criteria.

The top-k query answering problem has drawn a great deal of attention for all data representations. In the field of XML documents, *TASM* [110], Top-k Approximate Subtree Matching, is one of the most effective approaches for identifying the k best approximate matches of query tree in document trees. For a given query tree, *TASM* returns the k subtrees of the documents tree (consisting on nodes of the documents with their descendants) that are closest to the query tree according to canonical tree edit distance, and without any predefined rules for relaxation on subtrees that are more likely to respond to the query, *i.e.* all document subtrees are considered as a candidate answer. Given the hypothesis, the most crucial issue is to be able to have a runtime cost linear in the size of the document and space complexity that does not depend on the document size. In the following, we present and discuss the TASM algorithm.

Definition 9 ([110] Top-k Approximate Subtree Matching problem). Let q (query) and T (document) be ordered labeled trees, $n = |T|$ be the size of T, T_i be the subtree of T originated from t_i, $\delta(q, T_{i_j})$ be a distance function between ordered labeled trees, and $k \leq n$ be an integer. A sequence of subtrees, $R = (T_{i_1}, T_{i_2}, ..., T_{i_k})$, is a top-$k$ ranking of the subtrees of the document T with respect to the query q iff

1. the ranking contains the k subtrees that are closest to the query: $\forall T_j \notin R : \delta(q, T_{i_k}) \leq \delta(q, T_j)$, and
2. the subtrees in the ranking are sorted by their distance to the query: $\forall 1 \leq j < k : \delta(q, T_j) \leq \delta(q, T_{i_{j+1}})$.

The top-k approximate subtree matching (*TASM*) problem is the problem of computing a top-k ranking of the subtrees of a document T with respect to a query q.

Most solutions that tackle the problem of approximate tree matching are based on exploiting a search space in the form of a recursion tree that maps the query nodes to the data tree nodes. They never construct the recursion tree entirely and use pruning methods to reduce the search space. However, the main challenge is to be able to prune large subtrees and perform the expensive tree edit distance computation on small subtrees only.

The high-level idea is to consider only independent subtrees, called candidate trees, that are within a given size threshold τ, *i.e.,* these candidate trees are not within other candidate trees. The key challenge is to keep in the memory buffer only nodes with their subtrees that are likely to be matches. One can distinguish two types of nodes: candidate nodes belonging to candidate subtrees and must be kept; and non-candidate nodes that are roots of subtrees but they are too large for the candidate set.

Given the growing size of documents, an effective filtering solution in terms of memory is not obvious. In the following, we discuss the prefix ring buffer that allows to prune all subtrees above a threshold in a single postorder scan of the document and use a look-ahead of only $O(\tau)$ nodes. To this end, the ring buffer, used by *TASM*, is characterized by its size, $b = \tau + 1$, the start pointer s, that points to the first position in the buffer, and the end pointer e, that points to the last position in the buffer. These two pointers allow checking the state of the ring buffer. It is empty iff $s = e$, and it is full iff $s = (e + 1)\%b$ (% is the modulo operator). In this structure, two operations are defined: (1) append a new node and (2) remove the leftmost subtree/node.

The ring buffer pruning of a postorder queue of a document T and an empty ring buffer of size $\tau + 1$ is as follows:

1. Dequeue nodes from the postorder queue and append them to a ring buffer until the ring buffer is full or the postorder queue is empty.
2. If the leftmost node of the ring buffer is a non-leaf, then remove it from the buffer, otherwise add the leftmost valid subtree to the candidate set and remove it from the buffer.
3. Go to 1) if the postorder queue is not empty; go to 2) if the postorder queue is empty but the ring buffer is not; otherwise terminate.

The strong point of the proposed algorithm is that there is an effective bound on the largest subtrees of a document that can be deemed matches to a given query.

Using the prefix ring buffer and computing tree edit distance on small subtrees only give rise to an efficient solution to the problem. The space complexity of this solution is independent of the document size and thus, scales well to XML documents that do not fit into memory. This solution requires $O(m^2n)$ time, and $O(m^2 + mk)$ space, with m is the size of the query, n is the size of the document, and k is the desired number of answers.

In *TASM*, the tree edit distance is used as a similarity metric, in which the cost model for the edit operations is defined as follows: an insertion (deletion) operation has a cost equal to the cost of an inserted (deleted) node. The cost of a relabeling operation is defined as the average between deletion and insertion operations. However, the use of this cost model is too restrictive since it does not differentiate between the cost of two main kinds of similarities, *i.e.,* structural and content similarities. In most cases of real-world applications, these kinds of similarity have not the same importance. Moreover, such a solution may be able to differentiate between the matching information from those which enrich the answers.

In the following, we introduce a new cost model in which the relabeling operation is considered as the succession of two operations, a deletion operation followed by an insertion operation. Moreover, delete leaf nodes, and their independent ancestors have no cost since they are considered as enrichment for the answer.

Definition 10. Let T and q be two ordered labeled trees, $t_i \in V_T$ be a tree node, $q_j \in V_q$ be a query node, and *cst* be the cost assigned to a given node. An insertion operation of a node t_i is denoted $(\epsilon \rightarrow t_i)$, a deletion operation of a node t_i is denoted $(t_i \rightarrow \epsilon)$, and a relabeling operation of a node q_j by the label of a node t_i is denoted $(q_j \rightarrow t_i)$. We denote by $leaf$ a leaf node and $IndAnc$ an independent ancestor of a deleted leaf node. The edit operation costs are defined by

$$\delta(q_j, t_i) = \begin{cases} 2 \cdot cst & \text{if } (q_j \longrightarrow t_i), \\ cst & \text{if } (\epsilon \longrightarrow t_i), \\ cst & \text{if } (t_i \longrightarrow \epsilon) \wedge (t_i \neq leaf \wedge t_i \neq IndAnc), \\ 0 & \text{if } (t_i \longrightarrow \epsilon) \wedge (t_i = leaf \vee t_i = IndAnc). \end{cases} \tag{6.1}$$

The tree edit distance between T and q is defined as the sum of the costs of all performed edit operations:

$$\delta(T,q) = \min_{e_1,\ldots,e_n \in \gamma(T,q)} \sum_{i=1}^{n} cst(e_i), \tag{6.2}$$

where $\gamma(T,q)$ is the set of edit operations, and cst represents the edit operation cost.

Now, we define the similarity function.

Definition 11. Let T be a document tree, q be a query tree, and $T_v \in T$ be a subtree in T rooted on v. The similarity function $\theta(T_v,q)$ is defined by

$$\theta(T_v,q) = \frac{1}{\delta(T_v,q)\cdot\left(1+\frac{|Rel_l|+|Ins_l|}{|leaf_q|}\right)}, \tag{6.3}$$

where $|Rel_l|$ is the number of relabeled leaves, $|Ins_l|$ is the number of inserted leaves and $|leaf_q|$ represents the number of leaves in q.

The similarity function considers subtrees (fragments) as content entities (information) delimited by hierarchical structures. Thus, the fragment with the highest similarity is evaluated as the most relevant. The relevance of a fragment is affected when a leaf node is relabeled or inserted. In other words, if an information change occurs or missing information is detected, the current fragment will be penalized. The corresponding fragment loses the relevance obtained by the tree edit distance δ and its accuracy becomes less important. Example 7 shows the difference between our similarity function and that used in *TASM*.

Example 7. We consider the example of Figure 6.2, the subtrees T_1, T_2, T_3, T_4 are the results of the query q for $k = 4$. Using the $TASM$'s similarity, the subtree T_1 is on top of the list, since it corresponds to the minimum number of edit operations $\delta(T_1,q)$, while T_4 is the last in the list ($\delta(T_1,q) = 3$, $\delta(T_2,q) = 4$, $\delta(T_3,q) = 5$, and $\delta(T_4,q) = 7$). One can check that this ranking is correct if only the structural similarity is considered. However, this ranking is not suitable when both structural and contents similarities are considered. Using the similarity function θ, the final ranking is $\{T_4, T_3, T_1, T_2\}$, where $\theta(T_4,q) = 1$, $\theta(T_3,q) = 0.15$, $\theta(T_1,q) = 0.12$ and $\theta(T_2,q) = 0.086$).

Actually, with the growing size of data, information become scattered over multiple documents. Thus, the retrieval process will be applied to all documents D_i of the

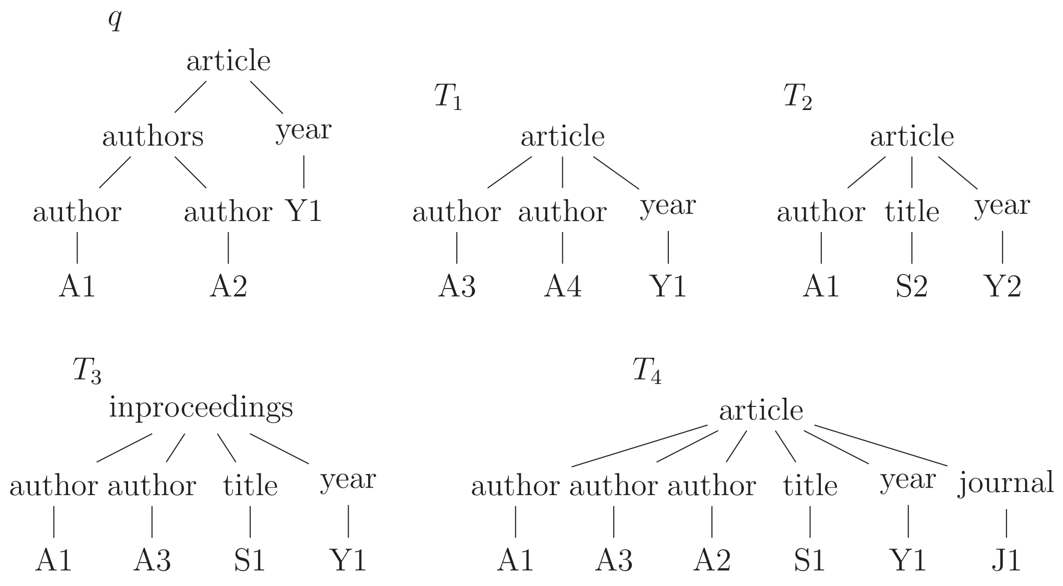


FIGURE 6.2: Example of query and its relevant subtrees returned by top-k approximate subtree matching algorithm, with $k = 4$.

collection C, and the result will be an ordered list L of target trees (fragments) of at most k items, for each document being queried.

It is clear that these fragments are heterogeneous, each having a different granularity. Filtering these lists to eliminate the irrelevant fragments is a necessary task. The filtering aims at evaluating each candidate fragment separately with structural and semantical (content) criteria. Let q be a query and f_i be a candidate fragment. The fragment f_i will be evaluated as irrelevant, and it will not be considered as a deemed response if it verifies the following condition:

$$(\delta(f_i, q) \geq (2 \cdot |q|)) \quad \vee \quad \left(\frac{|Rel_l| + |Ins_l|}{|leaf_q|} \geq \mu \right). \tag{6.4}$$

The left side of the condition is used to structurally distinguish q and f_i (indeed $2 \cdot |q|$ is reached if all the elements of q are relabeled). On the right side of the condition, $(0 \leq \mu \leq 1)$ is a user-defined parameter that presents an evaluation threshold for the fragment's content. If the number of inserted and relabeled leaves is close to the number of leaves in the query, then the fragment is considered irrelevant.

6.3.2 Aggregation process

Recall that the previous step generates, for each document in the collection, a list of fragments answering to the query q, ordered by the similarity function given above. In this section, we introduce and describe the aggregation process in order to construct one final result. The resulting document, called aggregate, should be as exhaustive and relevant as possible. It must be semantically correct with respect to the returned responses and should not contain redundancy either. This aggregation process is decomposed into different steps:

1. Selection of the fragments to be aggregated. At the first iteration, a base fragment f_{base} is selected. Then for each iteration, we identify, from the response lists, a fragment f to be aggregated to f_{base}. This identification is made among the lists not yet used.

2. Construction of the primary aggregate. We introduce the composition operators to aggregate f with f_{base}.

3. If the final aggregate is not yet obtained, the primary aggregate is considered as the new base fragment, and the process is repeated from step 1.

6.3.2.1 Step 1: Selection.

The selection process aims at selecting the fragments to be aggregated. In the first iteration, it also identifies the base fragment. For each iteration, it selects a fragment, from the list of answers L, that has the most additional relevant information to f_{base}. Since the most interesting fragment in L is already considered, we assume that no further fragment from L has relevant information and we remove L from the remaining of the process.

Base fragment: Among all the fragments of the optimized lists, which contain only relevant fragments, a fragment is chosen to be the aggregation base. This should be as relevant as possible to give a partial view of the aggregated result structure. Thus, the base fragment f_{base} is the fragment with the highest similarity among the best fragments of all the response lists.

Fragment selection: The selection of a fragment f is based on the comparison between f_{base} and all the fragments of response lists using a path decomposition of each tree (fragment). A path decomposition of a tree T is the set of paths from the root to each of its leaves (note that we keep the notations $root$ and $leaf$ for the first and last nodes of each path). Then, for each fragment f we construct a similarity matrix between f_{base} and f. We denote by $M(f_{base}, f)$ the similarity matrix of size $n = \max\{|f_{base}|, |f|\}$, where the rows of $M(f_{base}, f)$ represent the paths p_i of f_{base} and the columns represent the paths f_j of f. The matrix elements are computed as follows:

$$M(p_i, f_j) = \begin{cases} lcs(p_i, f_j) & \text{if } leaf(p_i) = leaf(f_j), \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

The *longest common subsequence* between two paths, denoted $lcs(p, p')$, is computed by the algorithm given in [258] which has $O(ND)$ time and space complexity, where N is the sum of the lengths of p and p' and D is the size of the minimum edit operations for p and p'. After constructing the similarity matrix, we use the Hungarian algorithm [57] to compute the matching on it. The Hungarian algorithm is considered as one of the most efficient matrix-matching algorithms with an $O(n^3)$ time complexity. The fragment for which the matching is maximized is selected for the next step.

6.3.2.2 Step 2: Construction of the aggregate.

The Hungarian algorithm gives the set of matchings between the path decompositions of f_{base} and f. Note that the information given by the paths (on their leaves) intersect, *i.e.* they are similar. Let $\mathcal{R} = \{(p_i, f_j)\}$ be the set of these matchings where $p_i \in f_{base}$, $f_j \in f$ and $lcs(p_i, f_j) \geq 1$. We consider each pair (p_i, f_j) independently, and for each pair (p_i, f_j) we define a subtree of f to be aggregated to f_{base}. Thus for each pair (p_i, f_j):

- The *inclusion subtree* $t(f_j)$ of f is the subtree of f rooted on the first ancestor that does not carry two intersection information. To do that we identify each node of f in a post-order. Thus $lcs(p_i, f_j)$ can be represented by the identifiers of its nodes. Let L be the list of identifiers of the $lcs(p_i, f_j)$ nodes. We also define R as the list of identifiers of all the leaves in $\mathcal{R} \backslash \{(p_i, f_j)\}$.

The root's identifier of the inclusion subtree is then given by $id = \max\{l \in L | \forall r \in R, r > l\}$.

- The composition of this inclusion subtree with f_{base} is done by merging the common nodes and by adding the new nodes of $t(f_j)$ in f_{base} (connected as in $t(f_j)$).
- We do $f = f \backslash \{t(f_j)\}$. We remove the subtree $t(f_j)$ from the fragment f before the new pair (p_i, f_j) (to avoid duplication of information in the next iterations). Note that f stays a connected tree after the deletion.

Finally if the fragment f still contains information (after considering all pairs of $\mathcal{R}$), the remaining tree f is added to f_{base} by adding a fictive node as root between f_{base} and f.

6.3.3 Complexity

In this section we analyze the running time of our method. Let q be a query of size $|q| = m$. A document D, of size $|D| = n$, is splitted into p documents $D_1, D_2, \ldots, D_p$. The computation of the time complexity can be divided into two parts. The first one is the complexity of the retrieval process and the second one is the complexity of the aggregation phase. The retrieval process is based on the TASM algorithm on p documents. Authors prove in [110] that the runtime complexity of TASM is $O(m^2 n)$ for a document. Thus for the entire part, the complexity of the retrieval process is $O(pm^2 n)$. In the other part, it is important to note that the number of the returned fragments for each D_i is k (with $k \leq n$) and the size of a returned fragment is bounded by $(2m + k)$ (by [110]). Since the computation of the similarity function is done in the process of retrieving, the filtering step is done in $O(pk)$ time. Moreover, the base fragment f_{base} is chosen in a constant time. Then for all the remaining D_i, all their fragments are considered. In the fragment selection, the cost to construct the similarity matrix is $O(m^2)$ and the cost of the Hungarian matching is $O(m^3)$. For each selected fragment, the inclusion subtree is computed in $O(m^2)$ and the composition (insertion, union, deletion) in $O(n)$. Thus for all the fragments of all documents, the aggregation phase is done in $O(p^2 k m^3)$ time. We can remark that the complexity is independent of the size of documents. It depends on the number of partitions p and the size of the query

m. Note that typically m is small and the size of documents is much larger than the size of query ($n >> m$).

6.4 Experiments

The evaluation of the aggregated search is a difficult challenge; it can be considered as an issue that is so far from being solved [21]. It may be related to several criteria like the complementarity of the returned elements that constitute the final aggregate, the non-redundancy of information and completeness.

In this section, we describe and discuss the experimental results to evaluate our proposed method. However, to our knowledge, there is no other method for aggregation based on structural and semantical constraints. Thus we compare our method with the *TASM* algorithm.

6.4.1 Test collection

Our evaluation is based on a test collection decomposed into three parts: a set of documents to be interrogated, a set of queries to be searched in the document collection and associated relevance judgments for each query. In our study, we construct the collections of documents from two real-world datasets (with XML format): the DBLP[2] dataset, which is an XML representation of bibliographic information on major computer science journals and proceedings; and the IMDb[3] dataset, which gives a representation of Internet Movie Database in an XML document. For each dataset, the whole document (*i.e., global document*) D_b is partitioned into p partitions that verify: $\bigcup_{i=1}^{p} P_i = D_b$ and $\bigcap_{i=1}^{p} P_i \neq \emptyset$. Each partition is considered as an independent document. In this work, queries are also given with an XML format, and for each dataset we construct a set of queries from the global document. Each set contains 20 queries with different sizes, structures and contents. Each of them is evaluated by a set of experts (researchers) to construct the relevant judgment sets that gather all relevant answers from the global document to a given query.

[2]http://dblp.uni-trier.de/xml/
[3]https://www.imdb.com/interfaces/

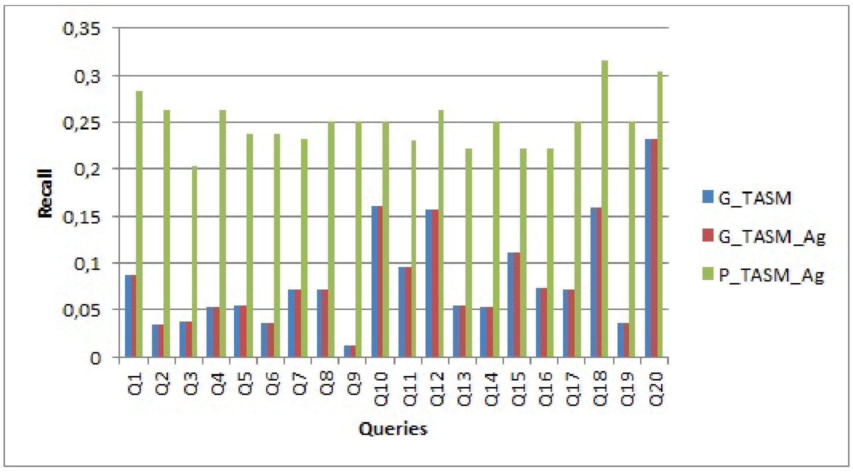


FIGURE 6.3: Recall for the three algorithms on IMDb dataset.

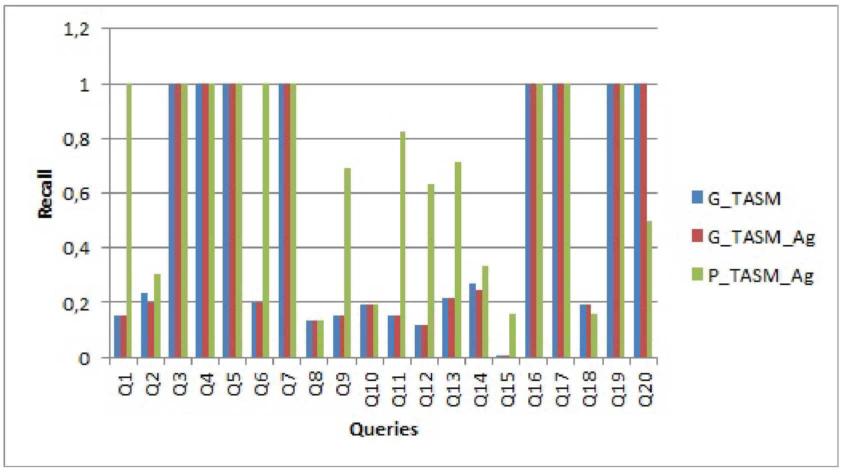


FIGURE 6.4: Recall for the three algorithms on DBLP dataset.

6.4.2 Evaluation method

We conduct three sets of experiments. The first one, called *Global TASM*, is an implementation of TASM with its cost model on the global documents D_b. The *TASM* algorithm returns for each document a list of k fragments that are deemed relevant by the tree edit distance function. The second one, called *Global TASM Aggregation*, is an implementation of the *TASM* algorithm with our cost model on the global documents D_b, augmented by the proposed aggregation approach applied on the list of fragments sorted by the similarity function $\theta()$. The last one called, *Partition TASM Aggregation*, is an implementation of our algorithm on the p partitions. In our experiments, we fix $\mu = 0.7$ and we evaluate our approach with several values of k varying from 1 to 15.

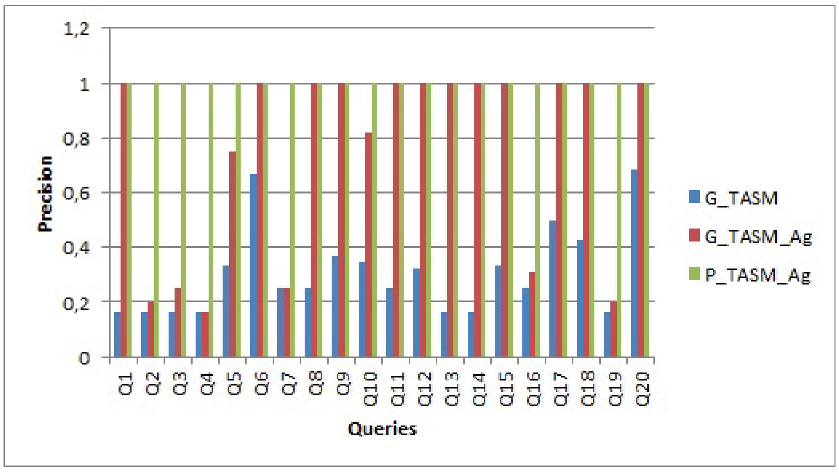


FIGURE 6.5: Precision for the three algorithms on IMDb dataset.

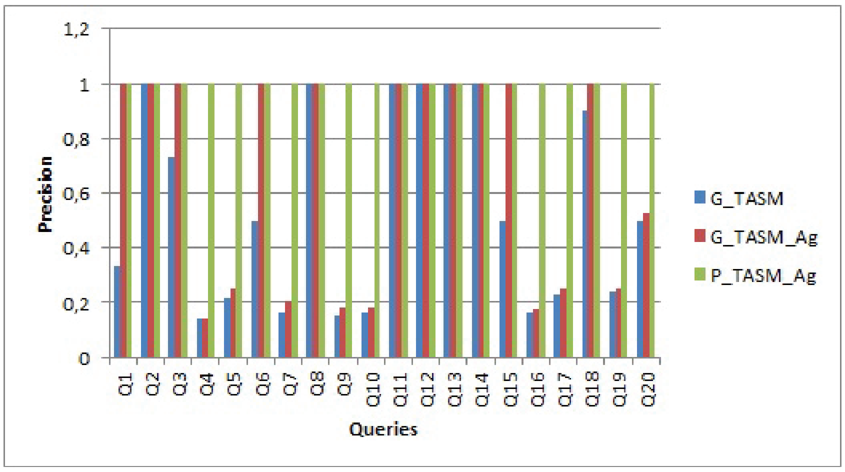


FIGURE 6.6: Precision for the three algorithms on DBLP dataset.

6.4.3 Result discussions

To evaluate the results of each experiment, we use two criteria. The first one is the *recall*, which represents the ratio between the number of relevant returned elements and the number of relevant elements of the response in the judgment set. The second one is the *precision*, which represents the ratio between the relevant returned elements and the number of returned elements.

Figures 6.3-6.4 show the recall of responses for the three algorithms on IMDb and DBLP datasets, respectively. They report the overall performance of our method in terms of relevant responses. In the IMDb dataset, *Partition TASM Aggregation* carries 30% (for *Q*20) to 90% (for *Q*9) more relevant information than both *Global TASM* and *Global TASM Aggregation*. In contrast, there is no gain in terms of response enrichment for *Global TASM Aggregation*. This is due to the fact that all the returned fragments (using *Global TASM* and *Global TASM*

Aggregation) are small since they are related to the query size and the value k, while the subtrees in the global document are large. We can consider the queries used to interrogate this dataset as aggregative queries, especially when the result is obtained by interrogating a set of documents to assemble relevant fragments. Each fragment contributes partially to the response, but all together constitute a complete response. The same results appear on the DBLP dataset for aggregative queries ($Q1, Q2, Q6, Q9, Q11-Q15$). However, the experiments show less efficiency compared to IMDb for the other queries. This is due to the nature of the queries that do not have a response carrying a gain in terms of information enrichment. Such queries are very generic and their judgment response sets are huge, which leads to considering all responses of *Global TASM* and *Global TASM Aggregation* as relevant. On the other hand, the results of these queries in *Partition TASM Aggregation* are the construction and enrichment of one fragment, which makes the corresponding recall small compared to the size of the judgment responses.

These results are evaluated without considering whether a response contains irrelevant and/or redundant elements. The redundancy of elements in the same response is considered as irrelevant information. For each query's answer, we consider the precision of returned elements to evaluate their qualities. Figures 6.5-6.6 show the precision of returned elements of the three algorithms on IMDb and DBLP datasets, respectively. In both IMDb and DBLP, *Partition TASM Aggregation* shows a high precision of results for the majority of queries. It returns only the relevant elements and eliminates redundancies. On the other hand, *Global TASM Aggregation* shows an improvement in terms of precision compared to *global TASM*. This is due to the elimination of redundancies of fragments returned.

6.5 Conclusion

In this chapter, we discussed a novel framework for approximate XML retrieval based on aggregated search. Firstly, we presented the aggregated tree search starting from motivation and general definition in the context of XML search. Then, we introduced a search model based on an approximate tree matching algorithm, the Top-k Approximate Subtree Matching, which was adapted to distinguish between structural and content similarities. This search model provides a set of fragments that candidate to answer a query. After that, we introduced an aggregation model that generates a single result document containing the most relevant elements and their complementary information. Our method showed its performance in terms of relevance and quality for the XML fragment search where the information are distributed over several documents. Our method can be readily extended to support any data format having a tree representation.

Chapter 7

Conclusion and Perspectives

Contents

In this chapter, we first conclude this thesis with a summary of our contributions, in Section 7.1. We then describe several directions for future work, in Section 7.2.

7.1 Conclusion

It has been recognized that graphs gain popularity constituting an integral part of many real-world applications. Due to the importance of graphs, many studies have invested heavily in data graph exploitation technologies. These efforts have resulted in an increasing number of graph querying approaches. Therefore, improving the capabilities of these approaches with effective and efficient techniques for graph search and selection becomes an important issue.

In this thesis, we addressed two important problems arising in the graph theory field. First, we tackled the problem of graph pattern matching, specifically for massive data. Second, we investigated the aggregated tree search problem, which consists in retrieving information from several data sources and aggregating them to provide a more complete and significant answer.

In the first part of this thesis, we addressed the (diversified) top-k graph pattern matching problem. Basically, this problem consists of determining equality between two (sub)graphs in terms of structure and labels. It also finds top-ranked answers that are close to the query graph based on given criteria. This problem is essential for a wide range of emerging applications and has been a research focus in industrial and academic fields. For example, it is usually used in community discovery, social network analysis, biological data analysis, web site classification, among other things. In this context, we realized mainly two major contributions. The first one is the introducing of a new graph pattern matching model, called *Relaxed Graph Simulation (RGS)*, allowing the relaxation of queries to identify meaningful matches and to avoid the empty-set answer problem. The second contribution in this part addresses the top-k and diversified top-k problems. We studied two classes of functions for ranking matches, relevance and diversity. Based on both of them, we proposed a diversification function to balance relevance and diversity. Furthermore, we provided efficient algorithms based on optimization strategies to compute the top-k and the diversified top-k matches according to the *RGS* model. We analyzed the time complexity of the proposed algorithms and conducted extensive experiments on real-life datasets, which attested the effectiveness of the proposed approach.

In the second part of this thesis, we addressed the aggregated search problem, more precisely the aggregated tree search problem. This problem aims to answer queries on several documents and provides a coherent final answer that contains the most relevant elements and their complementary information so as to cover the expectations of the query. We introduced and developed a new approach for aggregated tree search and we analyzed the theoretical and experimental performances on different real datasets.

7.2 Further works

This thesis leads to many fertile grounds for future research. We identify the following main directions for future work:

- The actual sheer increase in data has led to a plethora of graph datasets. For example, Facebook amounts to 1.562 million daily active users [9] and Twitter totals up to 330 million active users [10]. Moreover, these data are often partitioned and distributed, which has attracted the interest in creating new frameworks to query big graphs, *e.g.*, Graphlab [259], Pregel [260] and its open-source version Giraph [261]. Currently, we are engaged in data distribution and parallel execution across a cluster of machines in order to manage and exploit big data graph that do not fit into main memory. Our proposals are being extended to support the searching in distributed graphs. Our aim is to design algorithms that are parallel scalable in response time and data shipment. In other words, an algorithm is said to be parallel scalable in response time if the computational cost is determined only by the size of the query and the largest fragment(partition) of the graph; and in data shipment if the total amount shipped is decided by the size of the query and the number of fragments (partitions). For this purpose, we use the notion of distributed data graphs, which is defined as a fragmentation $\mathcal{F} = \{F_1, ..., F_n\}$ of $G = (V, E, l, \Sigma)$, where each fragment F_i is characterized by $(V_i \cup Fi.O, E_i, l, \Sigma)$ such that (1) V_i is a partition of V, (2) $Fi.O$ is the set of nodes v such that there exists a crossing edge $e = (u, v) \in E$, $u \in V_i$ is a local node and $v \in F_j$, with $i \neq j$, is a virtual node; and (3) $(V_i \cup Fi.O, E_i, l, \Sigma)$ is subgraph of G induced by $V_i \cup Fi.O$. Using the notion of distributed data graphs, we investigate the distributed graph pattern matching based on graph simulation and relaxed graph simulation. Furthermore, we are developing algorithms that combine both partial evaluations in each fragment and message passing. Besides, we study the combination of distributed processing with graph compression and (diversified) top-k query answering.

- In the aggregated tree search approach, the retrieval process is based on a top-k approximate subtree matching algorithm. The returned results are ranked by a relevance function that distinguishes between structural and content similarities. An interesting future direction is to develop algorithms and techniques that can benefit both from the relaxation principle proposed in the first part of the thesis and from a ranking function that can combine relevance, diversity and novelty of results, *i.e.,* the returned results are similar to the query, dissimilar to each other and contain new information different from the ones previously retrieved in order to improve the user's query satisfaction. In addition, this problem becomes more challenging when considering the complex representation of data in graph form. However, it is more difficult for distributed graphs.

Bibliography

[1] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

[2] Karen Sparck Jones. *Readings in information retrieval.* Morgan Kaufmann, 1997.

[3] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*, 2014.

[4] Terry L Friesz, Javier Luque, Roger L Tobin, and Byung-Wook Wie. Dynamic network traffic assignment considered as a continuous time optimal control problem. *Operations Research*, 37(6):893–901, 1989.

[5] Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[6] Brian P Kelley, Bingbing Yuan, Fran Lewitter, Roded Sharan, Brent R Stockwell, and Trey Ideker. Pathblast: a tool for alignment of protein interaction networks. *Nucleic acids research*, 32(suppl_2):W83–W88, 2004.

[7] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[8] Monika Rauch Henzinger, Thomas A Henzinger, and Peter W Kopke. Computing simulations on finite and infinite graphs. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 453–462. IEEE, 1995.

[9] Statista. Facebook: number of daily active users worldwide 2011-2019. https://www.statista.com/statistics/346167/facebook-global-dau/, july 2019.

[10] Statista. Twitter: number of monthly active users 2010-2019. https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/, july
2019.

[11] David S Johnson and Michael R Garey. *Computers and intractability: A guide to the theory of NP-completeness*, volume 1. WH Freeman San Francisco, 1979.

[12] Wenfei Fan. Graph pattern matching revised for social network analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21. ACM, 2012.

[13] Abdelmalek Habi, Brice Effantin, and Hamamache Kheddouci. Fast top-k search with relaxed graph simulation. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 495–502. IEEE, 2018.

[14] Gang Gou and Rada Chirkova. Efficient algorithms for exact ranked twig-pattern matching over graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 581–594. ACM, 2008.

[15] Marina Drosou and Evaggelia Pitoura. Disc diversity: result diversification based on dissimilarity and coverage. *Proceedings of the VLDB Endowment*, 6(1):13–24, 2012.

[16] Christopher Manning, Prabhakar Raghavan, and Hinrich Schütze. Introduction to information retrieval. *Natural Language Engineering*, 16 (1):100–103, 2010.

[17] W3C XML web page. http://www.w3.org/XML/.

[18] Mohammed Amin Tahraoui, Karen Pinel-Sauvagnat, Cyril Laitang, Mohand Boughanem, Hamamache Kheddouci, and Lei Ning. A survey on tree matching and xml retrieval. *Computer Science Review*, 8:1–23, 2013.

[19] Karen Spärck-Jones, Stephen E Robertson, and Mark Sanderson. Ambiguous requests: implications for retrieval tests, systems and theories. In *ACM SIGIR Forum*, volume 41, pages 8–17. ACM, 2007.

[20] Vanessa Murdock and Mounia Lalmas. Workshop on aggregated search. In *ACM SIGIR Forum*, volume 42, pages 80–83. ACM, 2008.

[21] Arlind Kopliku, Karen Pinel-Sauvagnat, and Mohand Boughanem. Aggregated search: A new information retrieval paradigm. *ACM Computing Surveys (CSUR)*, 46(3):41, 2014.

[22] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*, volume 290. Citeseer, 1976.

[23] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, NJ, 1996.

[24] Reinhard Diestel. Graph theory, volume 173 of. *Graduate texts in mathematics*, page 7, 2012.

[25] David E Ghahraman, Andrew KC Wong, and Tung Au. Graph optimal monomorphism algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(4):181–188, 1980.

[26] Andrew KC Wong, Manlai You, and SC Chan. An algorithm for graph optimal monomorphism. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(3):628–638, 1990.

[27] Horst Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

[28] John E Hopcroft and Jin-Kue Wong. Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 172–184. ACM, 1974.

[29] Alfred V Aho and John E Hopcroft. *The design and analysis of computer algorithms*. Pearson Education India, 1974.

[30] László Babai. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 684–697. ACM, 2016.

[31] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, Francesco Tortorella, and Mario Vento. Graph matching: a fast algorithm and its evaluation. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170)*, volume 2, pages 1582–1584. IEEE, 1998.

[32] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE transactions on pattern analysis and machine intelligence*, 26(10): 1367–1372, 2004.

[33] Peixiang Zhao and Jiawei Han. On graph query optimization in large networks. *Proceedings of the VLDB Endowment*, 3(1-2):340–351, 2010.

[34] Shijie Zhang, Shirong Li, and Jiong Yang. Gaddi: distance index based subgraph matching in biological networks. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 192–203. ACM, 2009.

[35] Julian R Ullmann. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *Journal of Experimental Algorithmics (JEA)*, 15:1–6, 2010.

[36] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. Turbo iso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 337–348. ACM, 2013.

[37] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1199–1214. ACM, 2016.

[38] Brendan D McKay et al. *Practical graph isomorphism.* Department of Computer Science, Vanderbilt University Tennessee, USA, 1981.

[39] Marco Gori, Marco Maggini, and Lorenzo Sarti. Exact and approximate graph matching using random walks. *IEEE transactions on pattern analysis and machine intelligence*, 27(7):1100–1111, 2005.

[40] Peter J Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. Matching graphs with unique node labels. *Pattern Analysis and Applications*, 7(3): 243–254, 2004.

[41] S Greenblatt, S Marcus, and T Darr. Tmods-integrated fusion dashboard-applying fusion of fusion systems to counter-terrorism. In *Proc. International Conference on Intelligence Analysis*, 2005.

[42] Nicholas Dahm, Horst Bunke, Terry Caelli, and Yongsheng Gao. Topological features and iterative node elimination for speeding up subgraph isomorphism detection. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1164–1167. IEEE, 2012.

[43] Surapong Auwatanamongkol. Inexact graph matching using a genetic algorithm for image recognition. *Pattern Recognition Letters*, 28(12): 1428–1437, 2007.

[44] Horst Bunke. Error correcting graph matching: On the influence of the underlying cost function. *IEEE transactions on pattern analysis and machine intelligence*, 21(9):917–922, 1999.

[45] Horst Bunke. Error-tolerant graph matching: a formal framework and algorithms. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 1–14. Springer, 1998.

[46] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3-4):255–259, 1998.

[47] Sébastien Sorlin, Christine Solnon, and Jean-Michel Jolion. A generic graph distance measure based on multivalent matchings. In *Applied Graph Theory in Computer Vision and Pattern Recognition*, pages 151–181. Springer, 2007.

[48] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[49] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *Proceedings of the VLDB Endowment*, 2(1):25–36, 2009.

[50] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Analysis and applications*, 13(1):113–129, 2010.

[51] Kaspar Riesen and Horst Bunke. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing*, 27(7):950–959, 2009.

[52] Kaspar Riesen, Miquel Ferrer, and Horst Bunke. Approximate graph edit distance in quadratic time. *IEEE/ACM transactions on computational biology and bioinformatics*, 2015.

[53] Michel Neuhaus and Horst Bunke. A quadratic programming approach to the graph edit distance problem. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 92–102. Springer, 2007.

[54] Sébastien Bougleux, Luc Brun, Vincenzo Carletti, Pasquale Foggia, Benoit Gaüzère, and Mario Vento. A quadratic assignment formulation of the graph edit distance. *arXiv preprint arXiv:1512.07494*, 2015.

[55] Francesc Serratosa. Computation of graph edit distance: Reasoning about optimality and speed-up. *Image and Vision Computing*, 40:38–48, 2015.

[56] Francesc Serratosa. Speeding up fast bipartite graph matching through a new cost matrix. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(02):1550010, 2015.

[57] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[58] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4): 325–340, 1987.

[59] Eugene L Lawler. The quadratic assignment problem. *Management science*, 9(4):586–599, 1963.

[60] Marius Leordeanu, Martial Hebert, and Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Advances in neural information processing systems*, pages 1114–1122, 2009.

[61] Peter E Hart, Nils J Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, (37):28–29, 1972.

[62] Wen-Hsiang Tsai and King-Sun Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on systems, man, and cybernetics*, 9(12):757–768, 1979.

[63] M Eshera and KS Fu. A similarity measure between attributed relational graphs for image analysis. In *Proc. 7th Int. Conf. Pattern Recognition*, pages 75–77, 1984.

[64] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. A look-ahead strategy for graph matching in retrieval by spatial arrangement. In *2000 IEEE International Conference on Multimedia and Expo. ICME2000. Proceedings. Latest Advances in the Fast Changing World of Multimedia (Cat. No. 00TH8532)*, volume 3, pages 1721–1724. IEEE, 2000.

[65] Stefano Berretti, Alberto Del Bimbo, and Enrico Vicario. Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10): 1089–1105, 2001.

[66] Michel Neuhaus, Kaspar Riesen, and Horst Bunke. Fast suboptimal algorithms for the computation of graph edit distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 163–172. Springer, 2006.

[67] Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, (1): 67–92, 1973.

[68] Josef Kittler and Edwin R Hancock. Combining evidence in probabilistic relaxation. *International Journal of Pattern Recognition and Artificial Intelligence*, 3(01):29–51, 1989.

[69] Richard C Wilson and Edwin R Hancock. Structural matching by discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):634–648, 1997.

[70] Yonathan Aflalo, Alexander Bronstein, and Ron Kimmel. On convex relaxation of graph isomorphism. *Proceedings of the National Academy of Sciences*, 112(10):2942–2947, 2015.

[71] HA Almohamad and Salih O Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE Transactions on pattern analysis and machine intelligence*, 15(5):522–525, 1993.

[72] Alan Chia-Lung Chen, Ahmed Elhajj, Shang Gao, Abdullah Sarhan, Salim Afra, Ahmad Kassem, and Reda Alhajj. Approximating the maximum common subgraph isomorphism problem with a weighted graph. *Knowledge-Based Systems*, 85:265–276, 2015.

[73] Swarup Medasani, Raghu Krishnapuram, and YoungSik Choi. Graph matching by relaxation of fuzzy assignments. *IEEE Transactions on Fuzzy Systems*, 9(1):173–182, 2001.

[74] Geoff Gross, Rakesh Nagi, and Kedar Sambhoos. A fuzzy graph matching approach in intelligence analysis and maintenance of continuous situational awareness. *Information Fusion*, 18:43–61, 2014.

[75] Brian Gallagher. Matching structure and semantics: A survey on graph-based pattern matching. *AAAI FS*, 6:45–53, 2006.

[76] Charu C Aggarwal, Haixun Wang, et al. *Managing and mining graph data*, volume 40. Springer, 2010.

[77] Junghoo Cho, Narayanan Shivakumar, and Hector Garcia-Molina. Finding replicated web collections. In *Acm Sigmod Record*, volume 29, pages 355–366. ACM, 2000.

[78] Lorenzo De Nardo, Francesco Ranzato, and Francesco Tapparo. The subgraph similarity problem. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):748–749, 2008.

[79] Lei Zou, Lei Chen, and M Tamer Özsu. Distance-join: Pattern match query in a large graph database. *Proceedings of the VLDB Endowment*, 2 (1):886–897, 2009.

[80] Joel Brynielsson, Johanna Högberg, Lisa Kaati, Christian Mårtenson, and Pontus Svenson. Detecting social positions using simulation. In *2010 International Conference on Advances in Social Networks Analysis and Mining*, pages 48–55. IEEE, 2010.

[81] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. Graph pattern matching: from intractable to polynomial time. *Proceedings of the VLDB Endowment*, 3(1-2):264–275, 2010.

[82] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Yinghui Wu. Adding regular expressions to graph reachability and pattern queries. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 39–50. IEEE, 2011.

[83] Shuai Ma, Yang Cao, Wenfei Fan, Jinpeng Huai, and Tianyu Wo. Strong simulation: Capturing topology in graph pattern matching. *ACM Transactions on Database Systems (TODS)*, 39(1):4, 2014.

[84] Jianliang Gao, Ping Liu, Xuedan Kang, Lixia Zhang, and Jianxin Wang. Prs: Parallel relaxation simulation for massive graphs. *The Computer Journal*, 59(6):848–860, 2016.

[85] Nancy Buchan and Rachel Croson. The boundaries of trust: Own and others' actions in the us and china. *Journal of Economic Behavior & Organization*, 55(4):485–504, 2004.

[86] Wenfei Fan, Xin Wang, and Yinghui Wu. Querying big graphs within bounded resources. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 301–312. ACM, 2014.

[87] Wenfei Fan, Jianzhong Li, Xin Wang, and Yinghui Wu. Query preserving graph compression. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 157–168. ACM, 2012.

[88] Jia Li, Yang Cao, and Shuai Ma. Relaxing graph pattern matching with explanations. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1677–1686. ACM, 2017.

[89] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[90] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66 (4):614–656, 2003.

[91] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. 2000.

[92] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

[93] Ihab F Ilyas, Walid G Aref, and Ahmed K Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 13(3):207–221, 2004.

[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*, volume 1, pages 281–290, 2001.

[95] Min Xie, LV Lakshmanan, and Peter T Wood. Efficient rank join with aggregation constraints. *Proceedings of the VLDB Endowment*, 4(11): 1201–1212, 2011.

[96] Nikos Ntarmos, Ioannis Patlakas, and Peter Triantafillou. Rank join queries in nosql databases. *Proceedings of the VLDB Endowment*, 7(7): 493–504, 2014.

[97] Senjuti Basu Roy, Tina Eliassi-Rad, and Spiros Papadimitriou. Fast best-effort search on graphs with multiple attributes. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):755–768, 2014.

[98] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.

[99] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 16–27. ACM, 2003.

[100] Hao He, Haixun Wang, Jun Yang, and Philip S Yu. Blinks: ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 305–316. ACM, 2007.

[101] Yan Qi, K Selçuk Candan, and Maria Luisa Sapino. Sum-max monotonic ranked joins for evaluating top-k twig queries on weighted data graphs. In *Proceedings of the 33rd international conference on Very large data bases*, pages 507–518. VLDB Endowment, 2007.

[102] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. Natural language question answering over rdf: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 313–324. ACM, 2014.

[103] Martin Theobald, Holger Bast, Debapriyo Majumdar, Ralf Schenkel, and Gerhard Weikum. Topx: efficient and versatile top-k query processing for semistructured data. *The VLDB Journal—The International Journal on Very Large Data Bases*, 17(1):81–115, 2008.

[104] Lei Zou, Lei Chen, and Yansheng Lu. Top-k subgraph matching query in a large graph. In *Proceedings of the ACM first Ph. D. workshop in CIKM*, pages 139–146. ACM, 2007.

[105] Xiaofeng Ding, Jianhong Jia, Jiuyong Li, Jixue Liu, and Hai Jin. Top-k similarity matching in large graphs with attributes. In *International Conference on Database Systems for Advanced Applications*, pages 156–170. Springer, 2014.

[106] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st international conference on Very large data bases*, pages 505–516. VLDB Endowment, 2005.

[107] Nitin Agarwal, Magdiel Galan Oliveras, and Yi Chen. Approximate structural matching over ordered xml documents. In *11th International Database Engineering and Applications Symposium (IDEAS 2007)*, pages 54–62. IEEE, 2007.

[108] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, and Gerhard Weikum. Searching rdf graphs with sparql and keywords. *IEEE Data Eng. Bull.*, 33 (1):16–24, 2010.

[109] H Prufer. Neuerbeweiseinessatzesüberpermutationen. *Archiv fur Mathematik und Physik*, 27.

[110] N. Augsten, D. Barbosa, M. Böhlen, and T. Palpanas. Tasm: Top-k approximate subtree matching. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 353–364, March 2010. doi: 10.1109/ICDE.2010.5447905.

[111] Andreas Wagner, Thanh Tran Duc, Günter Ladwig, Andreas Harth, and Rudi Studer. Top-k linked data query processing. In *Extended Semantic Web Conference*, pages 56–71. Springer, 2012.

[112] Hanghang Tong, Christos Faloutsos, Brian Gallagher, and Tina Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 737–746. ACM, 2007.

[113] Arijit Khan, Nan Li, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty, and Shu Tao. Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 901–912. ACM, 2011.

[114] Jiefeng Cheng, Xianggang Zeng, and Jeffrey Xu Yu. Top-k graph pattern matching over large graphs. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 1033–1044. IEEE, 2013.

[115] Wenfei Fan, Xin Wang, and Yinghui Wu. Diversified top-k graph pattern matching. *Proceedings of the VLDB Endowment*, 6(13):1510–1521, 2013.

[116] Shengqi Yang, Fangqiu Han, Yinghui Wu, and Xifeng Yan. Fast top-k search in knowledge graphs. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 990–1001. IEEE, 2016.

[117] Xiaohuan Shan, Guangxiang Wang, Linlin Ding, Baoyan Song, and Yan Xu. Top-k subgraph query based on frequent structure in large-scale dynamic graphs. *IEEE Access*, 6:78471–78482, 2018.

[118] Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166. ACM, 2012.

[119] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In *Proceedings of the 18th international conference on World wide web*, pages 381–390. ACM, 2009.

[120] Kaiping Zheng, Hongzhi Wang, Zhixin Qi, Jianzhong Li, and Hong Gao. A survey of query result diversification. *Knowledge and Information Systems*, 51(1):1–36, 2017.

[121] Marina Drosou, HV Jagadish, Evaggelia Pitoura, and Julia Stoyanovich. Diversity in big data: A review. *Big data*, 5(2):73–84, 2017.

[122] Albert Angel and Nick Koudas. Efficient diversity-aware search. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 781–792. ACM, 2011.

[123] Elena Demidova, Peter Fankhauser, Xuan Zhou, and Wolfgang Nejdl. Divq: diversification for keyword search over structured databases. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 331–338. ACM, 2010.

[124] Marina Drosou and Evaggelia Pitoura. Search result diversification. *SIGMOD record*, 39(1):41–47, 2010.

[125] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*, pages 22–32. ACM, 2005.

[126] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proceedings of the second ACM international conference on web search and data mining*, pages 5–14. ACM, 2009.

[127] Davood Rafiei, Krishna Bharat, and Anand Shukla. Diversifying web search results. In *Proceedings of the 19th international conference on World wide web*, pages 781–790. ACM, 2010.

[128] Yi Zhang, Jamie Callan, and Thomas Minka. Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 81–88. ACM, 2002.

[129] Charles LA Clarke, Maheedhar Kolla, Gordon V Cormack, Olga Vechtomova, Azin Ashkan, Stefan Büttcher, and Ian MacKinnon. Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 659–666. ACM, 2008.

[130] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Diversifying top-k results. *Proceedings of the VLDB Endowment*, 5(11):1124–1135, 2012.

[131] Kostas Stefanidis, Marina Drosou, and Evaggelia Pitoura. Perk: personalized keyword search in relational databases through preferences. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 585–596. ACM, 2010.

[132] Dong Xin, Hong Cheng, Xifeng Yan, and Jiawei Han. Extracting redundancy-aware top-k patterns. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–453. ACM, 2006.

[133] Zhengwei Yang, Ada Wai-Chee Fu, and Ruifeng Liu. Diversified top-k subgraph querying in a large graph. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1167–1182. ACM, 2016.

[134] Huiping Liu, Cheqing Jin, Bin Yang, et al. Finding top-k shortest paths with diversity. *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[135] Yuanyuan Zhu, Jeffrey Xu Yu, Hong Cheng, and Lu Qin. Graph classification: a diversified discriminative feature selection approach. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 205–214. ACM, 2012.

[136] Hanghang Tong, Jingrui He, Zhen Wen, Ravi Konuru, and Ching-Yung Lin. Diversified ranking on large graphs: an optimization viewpoint. In

Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 1028–1036. ACM, 2011.

[137] Rong-Hua Li and Jeffery Xu Yu. Scalable diversified ranking on large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 25(9): 2133–2146, 2012.

[138] Filip Radlinski and Susan Dumais. Improving personalized web search using result diversification. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 691–692. ACM, 2006.

[139] Gabriele Capannini, Franco Maria Nardini, Raffaele Perego, and Fabrizio Silvestri. Efficient diversification of web search results. *Proceedings of the VLDB Endowment*, 4(7):451–459, 2011.

[140] Lisi Chen and Gao Cong. Diversity-aware top-k publish/subscribe for text stream. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 347–362. ACM, 2015.

[141] Marina Drosou and Evaggelia Pitoura. Dynamic diversification of continuous data. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 216–227. ACM, 2012.

[142] Marina Drosou and Evaggelia Pitoura. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering*, 26(5): 1102–1116, 2013.

[143] Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 585–594. ACM, 2011.

[144] Zhi Liang, Meng Xu, Maikun Teng, and Liwen Niu. Netalign: a web-based tool for comparison of protein interaction networks. *Bioinformatics*, 22 (17):2175–2177, 2006.

[145] Linton Freeman. The development of social network analysis. *A Study in the Sociology of Science*, 1, 2004.

[146] Abdelmalek Habi, Brice Effantin, and Hamamache Kheddouci. Diversified top-k search with relaxed graph simulation. *Social Network Analysis and Mining*, 9(1):55, 2019.

[147] Meredith Ringel Morris, Jaime Teevan, and Katrina Panovich. What do people ask their social networks, and why?: a survey study of status message q&a behavior. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1739–1748. ACM, 2010.

[148] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[149] Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001.

[150] Michael J Kuby. Programming models for facility dispersion: The p-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19 (4):315–329, 1987.

[151] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 75–88. ACM, 2014.

[152] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2):122–144, 2004.

[153] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[154] Marcos R Vieira, Humberto L Razente, Maria CN Barioni, Marios Hadjieleftheriou, Divesh Srivastava, Caetano Traina, and Vassilis J Tsotras. On query result diversification. In *2011 IEEE 27th International Conference on Data Engineering*, pages 1163–1174. IEEE, 2011.

[155] Yansong Feng and Mirella Lapata. Topic models for image annotation and text illustration. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for*

Computational Linguistics, pages 831–839. Association for Computational Linguistics, 2010.

[156] Marijn Koolen, Gabriella Kazai, and Nick Craswell. Wikipedia pages as entry points for book search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 44–53. ACM, 2009.

[157] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. Efficient query processing in geographic web search engines. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 277–288. ACM, 2006.

[158] Fernando Diaz. Integration of news content into web results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 182–191. ACM, 2009.

[159] Jaime Arguello et al. Aggregated search. *Foundations and Trends® in Information Retrieval*, 10(5):365–502, 2017.

[160] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR'94*, pages 232–241. Springer, 1994.

[161] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[162] Jay Michael Ponte and W Bruce Croft. *A language modeling approach to information retrieval*. PhD thesis, University of Massachusetts at Amherst, 1998.

[163] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: the view from here. *natural language engineering*, 7(4):275–300, 2001.

[164] Boris Katz, Gary Borchardt, and Sue Felshin. Syntactic and semantic decomposition strategies for question answering from multiple resources. In *Proceedings of the AAAI 2005 workshop on inference for textual question answering*, pages 35–41. AAAI Press Menlo Park, CA, 2005.

[165] Véronique Moriceau and Xavier Tannier. Fidji: using syntax for validating answers in multiple documents. *Information retrieval*, 13(5):507–533, 2010.

[166] David Carmel, Yoelle S Maarek, Matan Mandelbrod, Yosi Mass, and Aya Soffer. Searching xml documents via xml fragments. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 151–158. ACM, 2003.

[167] Jamie Callan. Distributed information retrieval. In *Advances in information retrieval*, pages 127–150. Springer, 2002.

[168] Thi Truong Avrahami, Lawrence Yau, Luo Si, and Jamie Callan. The fedlemur project: Federated search in the real world. *Journal of the American Society for Information Science and Technology*, 57(3):347–358, 2006.

[169] Aditya Pal and Jaya Kawale. Leveraging query associations in federated search. In *Proceedings of the SIGIR 2008 Workshop on Aggregated Search*, volume 3. Citeseer, 2008.

[170] Antonio Gulli and Alessio Signorini. Building an open source meta-search engine. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1004–1005. ACM, 2005.

[171] M Shokouhi and L Si. Federated information retrieval. *Foundations and Trends in Information Retrieval, Upcoming Issue*, 2011.

[172] Shanu Sushmita, Hideo Joho, and Mounia Lalmas. A task-based evaluation of an aggregated search interface. In *International Symposium on String Processing and Information Retrieval*, pages 322–333. Springer, 2009.

[173] Arlind Kopliku. *Approaches to implement and evaluate aggregated search*. PhD thesis, Université de Toulouse, Université Toulouse III-Paul Sabatier, 2011.

[174] Arlind Kopliku, Firas Damak, Karen Pinel-Sauvagnat, and Mohand Boughanem. Interest and evaluation of aggregated search. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 154–161. IEEE Computer Society, 2011.

[175] Mounia Lalmas. Aggregated search. In *Advanced Topics in Information Retrieval*, pages 109–123. Springer, 2011.

[176] Jaime Arguello, Fernando Diaz, Jamie Callan, and Jean-Francois Crespo. Sources of evidence for vertical selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 315–322. ACM, 2009.

[177] Ke Zhou, Ronan Cummins, Mounia Lalmas, and Joemon M Jose. Evaluating aggregated search pages. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 115–124. ACM, 2012.

[178] Cécile Paris, Stephen Wan, and Paul Thomas. Focused and aggregated search: a perspective from natural language generation. *Information Retrieval*, 13(5):434–459, 2010.

[179] Cécile Paris, Stephen Wan, Ross Wilkinson, and Mingfang Wu. Generating personal travel guides-and who wants them? In *International Conference on User Modeling*, pages 251–253. Springer, 2001.

[180] Christina Sauper and Regina Barzilay. Automatically generating wikipedia articles: A structure-aware approach. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 208–216. Association for Computational Linguistics, 2009.

[181] Hoa Trang Dang, Diane Kelly, and Jimmy J Lin. Overview of the trec 2007 question answering track. In *Trec*, volume 7, page 63, 2007.

[182] Sihem Amer-Yahia, Francesco Bonchi, Carlos Castillo, Esteban Feuerstein, Isabel Mendez-Diaz, and Paula Zabala. Composite retrieval of diverse and complementary bundles. *IEEE Transactions on Knowledge and Data Engineering*, 26(11):2662–2675, 2014.

[183] Horatiu Bota, Ke Zhou, Joemon M Jose, and Mounia Lalmas. Composite retrieval of heterogeneous web search. In *Proceedings of the 23rd international conference on World wide web*, pages 119–130. ACM, 2014.

[184] Horaţiu Bota, Ke Zhou, and Joemon J Jose. Exploring composite retrieval from the users' perspective. In *European Conference on Information Retrieval*, pages 13–24. Springer, 2015.

[185] Xifeng Yan, Philip S Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 766–777. ACM, 2005.

[186] James Cheng, Yiping Ke, Wilfred Ng, and An Lu. Fg-index: towards verification-free query processing on graph databases. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 857–872. ACM, 2007.

[187] Kristin Tufte and David Maier. Aggregation and accumulation of xml data. *IEEE Data Eng. Bull.*, 24(2):34–39, 2001.

[188] Kristin Tufte and David Maier. Merge as a lattice-join of xml documents. In *28th Intl. Conf. VLDB*, 2002.

[189] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[190] Daniele Porello and Ulle Endriss. Ontology merging as social choice: Judgment aggregation under the open world assumption. *Journal of Logic and Computation*, 24(6):1229–1249, 2012.

[191] Ulle Endriss and Umberto Grandi. Graph aggregation. *Artificial Intelligence*, 245:86–114, 2017.

[192] Fatma Zohra Bessai-Mechmache and Zaia Alimazighi. Aggregated search in xml documents. *Journal of Emerging Technologies in Web Intelligence*, 4(2):181–188, 2012.

[193] Najeh Naffakhi and Rim Faiz. Aggregated search in xml documents: What to retrieve? In *2012 International Conference on Information Technology and e-Services*, pages 1–6. IEEE, 2012.

[194] Daniele Porello and Ulle Endriss. Ontology merging as social choice. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 157–170. Springer, 2011.

[195] Krisztian Balog, Pavel Serdyukov, and Arjen P De Vries. Overview of the trec 2010 entity track. Technical report, NORWEGIAN UNIV OF SCIENCE AND TECHNOLOGY TRONDHEIM, 2010.

[196] Michael J Cafarella, Michele Banko, and Oren Etzioni. Relational web search. In *WWW Conference*, 2006.

[197] Arlind Kopliku, Mohand Boughanem, and Karen Pinel-Sauvagnat. Towards a framework for attribute retrieval. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 515–524. ACM, 2011.

[198] Enrique Alfonseca, Marius Pasca, and Enrique Robledo-Arnuncio. Acquisition of instance attributes via labeled and related instances. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 58–65. ACM, 2010.

[199] Fabian M Suchanek, Mauro Sozio, and Gerhard Weikum. Sofie: a self-organizing framework for information extraction. In *Proceedings of the 18th international conference on World wide web*, pages 631–640. ACM, 2009.

[200] Jonathan Robie, Chris Wilson, Lauren Wood, Arnaud Le Hors, Scott Isaacson, Ian Jacobs, Mike Champion, Robert S Sutor, Gavin Nicol, and Steven B Byrne. Document object model (DOM) level 1. W3C recommendation, W3C, October 1998. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/.

[201] Sihem Amer-Yahia and Mounia Lalmas. Xml search: languages, inex and scoring. *ACM SIGMOD Record*, 35(4):16–23, 2006.

[202] Don Chamberlin, Mary Fernandez, Jerome Simeon, Scott Boag, Jonathan Robie, Michael Kay, and Anders Berglund. XML path language (XPath) 2.0 (second edition). W3C recommendation, W3C, December 2010. http://www.w3.org/TR/2010/REC-xpath20-20101214/.

[203] Alessandro Campi, Ernesto Damiani, Sam Guinea, Stefania Marrara, Gabriella Pasi, and Paola Spoletini. A fuzzy extension of the xpath query language. *Journal of Intelligent Information Systems*, 33(3):285, 2009.

[204] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed extended xpath i (nexi). In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 16–40. Springer, 2004.

[205] Michael Dyck, Josh Spiegel, and Jonathan Robie. XQuery 3.1: An XML query language. W3C recommendation, W3C, March 2017. https://www.w3.org/TR/2017/REC-xquery-31-20170321/.

[206] Stephen Buxton, Michael Dyck, Sihem Amer-Yahia, Pat Case, Jim Melton, Mary Holstege, Michael Rys, Chavdar Botev, Jayavel Shanmugasundaram, and Jochen Dörre. XQuery and XPath full text 1.0. W3C recommendation, W3C, March 2011. http://www.w3.org/TR/2011/REC-xpath-full-text-10-20110317/.

[207] Christoph M Hoffmann and Michael J O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, 1982.

[208] Jochen Burghardt. A tree pattern matching algorithm with reasonable space requirements. In *Colloquium on Trees in Algebra and Programming*, pages 1–15. Springer, 1988.

[209] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[210] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3): 422–433, July 1979. ISSN 0004-5411. doi: 10.1145/322139.322143. URL http://doi.acm.org/10.1145/322139.322143.

[211] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[212] Philip Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.

[213] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[214] Philip N Klein. Computing the edit-distance between unrooted ordered trees. In *European Symposium on Algorithms*, pages 91–102. Springer, 1998.

[215] Mateusz Pawlik and Nikolaus Augsten. Rted: a robust algorithm for the tree edit distance. *Proceedings of the VLDB Endowment*, 5(4):334–345, 2011.

[216] Serge Dulucq and Hélène Touzet. Decomposition algorithms for the tree edit distance problem. *Journal of Discrete Algorithms*, 3(2-4):448–471, 2005.

[217] Pekka Kilpeläinen and Heikki Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing*, 24(2):340–356, 1995.

[218] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees—an alternative to tree edit. *Theoretical Computer Science*, 143(1):137–148, 1995.

[219] Mohammed Amin Tahraoui. *Coloring, packing and embedding of graphs*. PhD thesis, Université Claude Bernard-Lyon I, 2012.

[220] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On supporting containment queries in relational database management systems. In *ACM Sigmod Record*, volume 30, pages 425–436. ACM, 2001.

[221] Shurug Al-Khalifa, Hosagrahar V Jagadish, Nick Koudas, Jignesh M Patel, Divesh Srivastava, and Yuqing Wu. Structural joins: A primitive for efficient xml query pattern matching. In *Proceedings 18th International Conference on Data Engineering*, pages 141–152. IEEE, 2002.

[222] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic twig joins: optimal xml pattern matching. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 310–321. ACM, 2002.

[223] Jiaheng Lu, Ting Chen, and Tok Wang Ling. Efficient processing of xml twig patterns with parent child edges: a look-ahead approach. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 533–542. ACM, 2004.

[224] Ting Chen, Jiaheng Lu, and Tok Wang Ling. On boosting holism in xml twig pattern matching using structural indexing techniques. In *Proceedings*

of the 2005 ACM SIGMOD international conference on Management of data, pages 455–466. ACM, 2005.

[225] Guoliang Li, Jianhua Feng, Yong Zhang, and Lizhu Zhou. Efficient holistic twig joins in leaf-to-root combining with root-to-leaf way. In *International Conference on Database Systems for Advanced Applications*, pages 834–849. Springer, 2007.

[226] Junfeng Zhou, Min Xie, and Xiaofeng Meng. Twigstack+: Holistic twig join pruning using extended solution extension. *Wuhan University Journal of Natural Sciences*, 12(5):855–860, 2007.

[227] Zhewei Jiang, Cheng Luo, Wen-Chi Hou, Qiang Zhu, and Dunren Che. Efficient processing of xml twig pattern: A novel one-phase holistic solution. In *International Conference on Database and Expert Systems Applications*, pages 87–97. Springer, 2007.

[228] Songting Chen, Hua-Gang Li, Junichi Tatemura, Wang-Pin Hsiung, Divyakant Agrawal, and K Selçuk Candan. Twig 2 stack: bottom-up processing of generalized-tree-pattern queries over xml documents. In *Proceedings of the 32nd international conference on Very large data bases*, pages 283–294. VLDB Endowment, 2006.

[229] Jiaheng Lu, Tok Wang Ling, Chee-Yong Chan, and Ting Chen. From region encoding to extended dewey: On efficient processing of xml twig pattern matching. In *Proceedings of the 31st international conference on Very large data bases*, pages 193–204. VLDB Endowment, 2005.

[230] Jiaheng Lu, Xiaofeng Meng, and Tok Wang Ling. Indexing and querying xml using extended dewey labeling scheme. *Data & Knowledge Engineering*, 70(1):35–59, 2011.

[231] Pavel Zezula, Giuseppe Amato, Franca Debole, and Fausto Rabitti. Tree signatures for xml querying and navigation. In *International XML Database Symposium*, pages 149–163. Springer, 2003.

[232] Pavel Zezula, Federica Mandreoli, and Riccardo Martoglia. Tree signatures and unordered xml pattern matching. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 122–139. Springer, 2004.

[233] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S Yu. Vist: a dynamic index method for querying xml data by tree structures. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 110–121. ACM, 2003.

[234] Praveen Rao and Bongki Moon. Prix: Indexing and querying xml using prufer sequences. In *Proceedings. 20th International Conference on Data Engineering*, pages 288–299. IEEE, 2004.

[235] Haixun Wang and Xiaofeng Meng. On the sequencing of tree structures for xml indexing. In *21st International Conference on Data Engineering (ICDE'05)*, pages 372–383. IEEE, 2005.

[236] Sayyed Kamyar Izadi, Theo Härder, and Mostafa S Haghjoo. S3: Evaluation of tree-pattern xml queries supported by structural summaries. *Data & Knowledge Engineering*, 68(1):126–145, 2009.

[237] Xin Wu and Guiquan Liu. Xml twig pattern matching using version tree. *Data & Knowledge Engineering*, 64(3):580–599, 2008.

[238] Gang Gou and Rada Chirkova. Efficiently querying large xml data repositories: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(10):1381–1403, 2007.

[239] Marouane Hachicha and Jerome Darmont. A survey of xml tree patterns. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):29–46, 2011.

[240] Shtwai Alsubai. *Child Prime Label Approaches to Evaluate XML Structured Queries*. PhD thesis, University of Sheffield, 2018.

[241] Karen Sauvagnat, Lobna Hlaoua, and Mohand Boughanem. Xfirm at inex 2005: ad-hoc and relevance feedback tracks. In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 88–103. Springer, 2005.

[242] Norbert Fuhr, Mounia Lalmas, Saadia Malik, and Gabriella Kazai. *Advances in XML Information Retrieval and Evaluation: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl Castle, Germany, November 28-30, 2005. Revised and Selected Papers*, volume 3977. Springer, 2006.

[243] Norbert Fuhr, Mounia Lalmas, and Andrew Trotman. *Comparative Evaluation of XML Information Retrieval Systems: 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006 Dagstuhl Castle, Germany, December 17-20, 2006 Revised and Selected Papers*, volume 4518. Springer, 2007.

[244] Shlomo Geva, Jaap Kamps, and Andrew Trotman. *Advances in Focused Retrieval: 7th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2008, Dagstuhl Castle, Germany, December 15-18, 2009. Revised and Selected Papers*, volume 5631. Springer, 2009.

[245] Shlomo Geva, Jaap Kamps, and Andrew Trotman. *Comparative Evaluation of Focused Retrieval: 9th International Workshop of the Inititative for the Evaluation of XML Retrieval, INEX 2010, Vught, The Netherlands, December 13-15, 2010, The Netherlands, Revised Selected Papers*, volume 6932. Springer Science & Business Media, 2011.

[246] Giovanna Guerrini. Approximate xml query processing. In *Advanced Query Processing*, pages 129–155. Springer, 2013.

[247] Torsten Schlieder and Felix Naumann. Approximate tree embedding for querying xml data. 2000.

[248] Eugen Popovici, Gildas Ménier, and Pierre-François Marteau. Sirius: a lightweight xml indexing and approximate search system at inex 2005. In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 321–335. Springer, 2005.

[249] Raghav Kaushik, Rajasekar Krishnamurthy, Jeffrey F Naughton, and Raghu Ramakrishnan. On the integration of structure indexes and inverted lists. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 779–790. ACM, 2004.

[250] Mir Sadek Ali, Mariano Consens, Xin Gu, Yaron Kanza, Flavio Rizzolo, and Raquel Stasiu. Efficient, effective and flexible xml retrieval using summaries. In *International Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 89–103. Springer, 2006.

[251] Mai Dong Le and KP Sauvagnat. Utilisation de la distance d'édition pour l'appariement sémantique de documents xml. In *Proceedings of GAOC, Workshop*, 2010.

[252] Abdeslame ALILAOUAR and Florence SEDES. Fuzzy querying of xm l documents the minimum spanning tree. *OSWIR 2005*, page 11, 2005.

[253] A Alilaouar. Interrogation flexible de documents semi-structurés, 2007.

[254] Ronald L Graham and Pavol Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[255] Jian Liu and DL Yan. Answering approximate queries over xml data. *IEEE Transactions on Fuzzy Systems*, 24(2):288–305, 2015.

[256] Ismael Sanz, Marco Mesiti, Giovanna Guerrini, and Rafael Berlanga. Fragment-based approximate retrieval in highly heterogeneous xml collections. *Data & Knowledge Engineering*, 64(1):266–293, 2008.

[257] Abdelmalek Habi, Brice Effantin, and Hamamache Kheddouci. Search and aggregation in xml documents. In *International Conference on Database and Expert Systems Applications*, pages 290–304. Springer, 2017.

[258] Eugene W. Myers. Ano(nd) difference algorithm and its variations. *Algorithmica*, 1(1):251–266, 1986. ISSN 1432-0541. doi: 10.1007/BF01840446. URL http://dx.doi.org/10.1007/BF01840446.

[259] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[260] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[261] Ching Avery. Giraph: Large-scale graph processing infrastructure on hadoop. *Proceedings of the Hadoop Summit. Santa Clara*, 11(3):5–9, 2011.