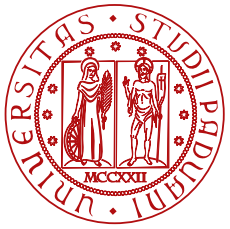

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Analysis of multiple update techniques on a RDF keyword search system

Master Candidate

Andrea Cassetta

Student ID 2019167

Supervisor

Prof. Gianmaria Silvello

University of Padova

Academic Year 2021/2022

12/12/2022

To infinity
...and beyond!

Abstract

Keyword search is a technology that allows non-expert users to explore and retrieve information and it is traditionally used for unstructured data, such as in Web page searches. In the last decade, this search method has also become popular for exploring structured data, such as relational databases or graphs. Instead of using complex SQL or SPARQL queries and when the underlying schema is known, the user writes a series of words(keywords) to search for what he or she needs, getting as answers the ones more matching with the search. Keyword search systems are challenged by two fundamental parameters, efficiency and effectiveness. In fact, efficiency and effectiveness are two qualities of a SPARQL, or SQL, query that returns an answer quickly and always accurate even when operating on large amounts of data. The "virtual documents" method allows keyword search systems to work also on large databases by generating answers to keyword queries in a reasonable time. This paper aims to replicate the keyword search systems based on "virtual documents" TSA+BM25 and TSA+VDP for RDF graphs. In addition, two methods of update processing in a keyword search system, will be presented and analyzed: BruteForce and semiTSA. Although keyword search is a growing research matter, the topic of updates on structured data, such as RDF data, had not yet been addressed in the literature.

Sommario

Il keyword search è una tecnologia che permette ad utenti non esperti di esplorare e ricavare informazioni ed è tradizionalmente usata per dati non strutturati, come nelle ricerca di pagine Web. Nellultima decade questo metodo di ricerca ha preso piede anche per esplorare dati strutturati, come database relazionali o grafi. Invece di usare complesse query SQL o SPARQL e dovendone conoscere lo schema sottostante, lutente scrive una serie di parole (keywords) per cercare ciò di cui ha bisogno ottenendo delle risposte il più possibili aderenti alla ricerca. I sistemi di keyword search devono confrontarsi con due parametri fondamentali: efficienza e efficacia. Efficienza ed efficacia infatti sono due qualità che consentono ad una query SPARQL, o SQL, di restituire una risposta in modo veloce e sempre preciso anche operando su grandi quantità di dati. Il metodo dei "documenti virtuali" permette ai sistemi di keyword search di lavorare anche su database di grandi dimensioni generando risposte alle keyword query in un tempo ragionevole. Questo lavoro ha lobiettivo di replicare i sistemi di keyword search basati su "documenti virtuali" TSA+BM25 e TSA+VDP per grafi RDF. Saranno, inoltre, presentati e analizzati due metodi di gestione degli aggiornamenti in un sistema di keyword search: BruteForce e semiTSA. Pur essendo il keyword search un argomento di ricerca in espansione, il tema degli aggiornamenti su dati strutturati, come i dati RDF, non era ancora stato affrontato in letteratura

Contents

1 Introduction **1**

2 Background **5**

2.1 RDF 5

2.1.1 SPARQL 6

2.2 Keyword search on structured data 6

2.3 TSA - Topological Syntactical Aggregator 9

2.3.1 Offline phase 10

2.3.2 Online phase 11

2.3.3 Evaluation 13

3 The Replicability work **17**

3.1 Software and data 17

3.1.1 Data processing 17

3.2 The recipe 19

3.3 Testing 24

4 Update idea **27**

4.1 Brute force approach 28

4.2 Semi TSA approach 33

5 Analysis **39**

5.1 Update construction 40

5.2 Brute force strategy 40

5.3 Semi TSA strategy 43

5.3.1 Single update 43

5.4 Comparison between methods 45

5.4.1 Execution times 48

5.5 Multiple updates 63

6 Conclusions **67**

References **69**

List of Figures

2.1 Example of a dummy RDF graph provided by W3C community . 6
2.2 Running example of the TSA system, both BM25 and VDP pipelines are shown 10

3.1 Original schema upon which the LUBM data are generated. there are two distinct schemes 18
3.2 ER Schema of the IMDB dataset 19
3.3 Sample of a LinkedMDB entities 20
3.4 Schema of the TSA processes and elements the interact with . . . 21
3.5 Query example: here the information need is to find all Michael Bay's works, their type, title and genre 22
3.6 The answer (partial) to the query in Fig. 3.5, here the results is reported as an RDF graph in TURTLE format 23
3.7 Logic schema followed to reproduce the report data 23

4.1 Visual representation of how the dataset to be considered as an update was created. 27
4.2 Overview of the brute force approach to handle the update 28
4.3 An example of how the BruteForce method works on an entity such as Spike Lee 29
4.4 Overview of the Semi TSA approach to process the update 33
4.5 An example of how the semiTSA method works on an entity such as Spike Lee 34

5.1 Comparison between methods using BM25 in IMDB tb-DCG at 10% 50
5.2 Differences in tb-DCG between BruteForce and semi-TSA in IMDB 10% update using BM25 pipeline 51
5.3 Differences in tb-DCG between BruteForce and semi-TSA in IMDB 20% update using BM25 pipeline 51
5.4 Differences in tb-DCG between BruteForce and semi-TSA in IMDB 40% update using BM25 pipeline 52

5.5 Comparison between methods using BM25 in IMDB tb-DCG at 40% 53
5.6 Comparison between methods using VDP in IMDB tb-DCG at 10% 54
5.7 Differences in tb-DCG between BruteForce and semi-TSA in IMDB 10% update using VDP pipeline 55
5.8 Differences in tb-DCG between BruteForce and semi-TSA in IMDB 20% update using VDP pipeline 55
5.9 Differences in Recall between BruteForce and semi-TSA in IMDB 10% update using BM25 pipeline 56
5.10 Differences in Recall between BruteForce and semi-TSA in IMDB 20% update using BM25 pipeline 56
5.11 Differences in Recall between BruteForce and semi-TSA in IMDB 10% update using VDP pipeline 57
5.12 Comparison between methods using BM25 in IMDB tb-DCG at 10% 58
5.13 Differences in tb-DCG between BruteForce and semi-TSA in LinkedMDB 30% update using BM25 pipeline 59
5.14 Comparison between methods using VDP in LinkedMDB tb-DCG at 30% 60
5.15 Differences in tb-DCG between BruteForce and semi-TSA in LinkedMDB 30% update using VDP pipeline 61
5.16 Differences in recall between BruteForce and semi-TSA in LinkedMDB 10% update using BM25 pipeline 61
5.17 Differences in recall between BruteForce and semi-TSA in LinkedMDB 10% update using VDP pipeline 62
5.18 Processing time of the different update percentages between BruteForce and TSA 62
5.19 (a) tb-DCG results with multiple updates compared to a single one. *e.g. 20% means 5 updates of 4% each, and the other percentages follow the same pattern.* (b) Recall performance between a single and a multiple update. 64

List of Tables

3.1 Results obtained in my attempt to replicate the TSA+BM25 and TSA+VDP systems 24
3.2 Raw and percentage differences in my replicability work results. 24

5.1 Name of the dataset used and their dimension 40
5.2 Results obtained with the brute force approach 41
5.3 Results obtained with TSA pipelines and a variable update percentage 43
5.4 Offline time execution of the different systems 49
5.5 Results obtained with update split in 5 63

1

Introduction

Keyword search is traditionally associated with the image of Web search engines. Those engines offer the ability to explore a large amount of unstructured data easily, be it texts, web pages, or pictures. Writing some words, called query, to express the information need and having results in a matter of tenths of a second is an operation done daily by everyone. With unstructured data, we can refer to the texts of a blog or email messages of a company that therefore do not have a well-defined and standard structure. Structured data, on the contrary, have a well-defined schema on top of which data are organized. Usually, structured data are stored in relational databases, which is why this type of data is generally called relational data.

The keyword search has gained more and more traction to allow easier access to structured data to eliminate that barrier of knowledge and experience that was previously required. Traditionally, to search a relational database, a user must know the underlying schema in detail and create a query that meets its requirements. In recent years new structured data are gaining popularity driven by the growing mass of data produced by social networks and Web sites that allow the user to have a personalized experience. This data is usually stored in relational databases. Alongside these things has been the development of the Web of Data or semantic web in the form of RDF data.

RDF (Resource Description Framework) is a schema-free standard to represent knowledge by defining relationships between objects and creating a graph. In RDF, a concept can be expressed by a triple composed of three linked pieces: subject, predicate and object. An RDF graph is composed of a series of triples, joined like atoms. RDF allowed easier data sharing and integration thanks to the absence of a schema. This is why it is the most used data model in the knowledge graphs like DBPedia, Wikidata, DrugBank and OpenPHACTS. These are examples of repositories where data can come from different sources collected under a single roof where a schema can be present. Still, it is not always easy to

understand. When the schema is known, it becomes critical for writing robust and accurate SPARQL queries to explore the graph. A keyword search system on RDF data allows the user to describe his information need through a bag of words. A keyword query can be defined as a set $K = \{k_1, k_2, ..., k_n\}$ of keywords. Systems like DBXplorer, SPARK or the recent SQUIRREL rewrite the keywords into a SQL queries. While the keyword search problem on relational data is advantaged by the presence of an underlying schema in the RDF graph some additional challenges open up for developers. If a user wanted to know how many world titles Sebastian Vettel has won, he would write *"world titles won by Sebastian Vettel"*. Efficiency and effectiveness, together with correctly interpreting the user's intentions and understanding how the words are related, are the tasks to answer in the best possible way. Efficiency means that the answers must be given in a congruous time and space, while effectiveness measures how adherent to and relevant to the query the answers are. Dwelling on the result, there are two possible types: the exact-match and the best-match. The former is the paradigm of the precise and user-written SQL or SPARQL queries where the result is only one; the latter will present to the user a ranking of the documents, or requested objects, where the most relevant results of the query are placed, in the top positions, in a laddered fashion. We find many papers in the literature on the best-match approach. Recently, Dosso and Silvello[6] proposed two new keyword search systems based on the virtual document approach to face the keyword search problem on structured data, focusing on graph data. In a virtual document approach, already explored by others in [25],[22] and [9], that represents state of the art, the RDF graph is clustered, and then the single clusters are converted into text joining nodes and arcs with their metadata one after the other to form an indexable text.

Specifically, Dosso and Silvello[6] presented two systems, TSA+BM25 and TSA+VDP, where TSA stands for Topological Syntactical Aggregator. Both methods are divided into two parts, offline and online phases. In the offline part the RDF graph is analyzed to be divided into subgraphs following an accurate algorithm that tries to create clusters around a single topic(seed), like a movie if the dataset deals with cinema concepts. All the subgraphs are then converted into a text collection. This collection will be indexed using state-of-the-art techniques. While the offline part is common to both systems, the thing that distinguishes them is the online part. In the online phase the user's query comes into play (query expressed as a series of single terms). TSA+BM25 uses

well-known information retrieval techniques and it uses the BM25 function to sort the documents. These documents are then used to propose to the user a ranking of the subgraphs that answer the query. TSA+VDP, on the other hand, proposes a more articulated idea that starts from the very results of BM25. The subgraphs returned from BM25 are re-analyzed and cleaned of any incorrect information to produce a more accurate response. In that paper they also proposed a benchmark to evaluate keyword search systems on graphs to try to fill a gap that exists in the field. They created a keyword search system for graph data that scaled well and returned more accurate responses than state-of-the-art systems.

This work describes our aim to run that source code and then replicate the same experiments to get the same results. Therefore, we had to dive into their Java code to understand how the TSA algorithm was physically implemented, what auxiliary structures it relied on, and how the RDF graph was handled. Among the datasets used there were IMDB and LinkedMDB. The IMDB is a relational dataset, in fact it was transformed into an RDF graph. The LinkedMDB, on the other hand, is known to be a native RDF dataset. We were able to reproduce the whole system, slightly improving the final results. The replicability work was necessary to know deeply the two systems to study how the systems would react in the presence of any updates. In a structured system, an update is the insertion, modification, or removal of a piece of data. This can lead to some questions and challenges: how does this alteration affect the keyword search system's response? Does it slow down the mechanism? Does it worsen the end result? To what extent does the system function correctly? We looked for the answers in the literature. Unfortunately, it seems that published literature has no answers to these questions. So alongside the replicability work, we tried to investigate the topic of updates in a keyword search system over graph data by developing two different methods of handling them. In this work we considered as an update only the case of adding information. In detail, the update was simulated by dividing the graph into two parts, of varying sizes, e.g. 90/10 or 80/20 this means that 10 or 20 percent of the dataset triples were removed and then added later as if it were an actual update over the starting dataset.

The first method developed is called BruteForce. BruteForce is a very simple method because it considers the triples of the update individually, once the clustering operation is finished, which takes place in the first phase of the offline part. The triples one by one are inserted into the correct cluster where the subject

of the new triple and the cluster seed match. In the end, the original clusters will grow in size by having new triples and, consequently, new information. If a triple is not included in any cluster the system then proceeds with the TSA algorithm, which starts from the subject of that triple. The TSA algorithm, in this case, is confined to the dataset update. As the clustering is finished and the update dataset is totally explored, the process continues as in the original paper: the subgraphs are transformed into text and indexed, ready to accept user queries.

The second strategy to handle an update is called semi-TSA. This second method is based on the original algorithm and it follows the same logic by considering the "update" dataset as a separate entity. The TSA algorithm acts directly on this small dataset to create new clusters, in the same way they were created in the "base" dataset. At this stage the original metadata, identified in the "base" dataset, is preserved to improve the cluster in the update graph.

Because the changes were concentrated in the offline phase, the online part remained the same as in the original paper with both the BM25 and VDP pipelines handling the response to queries.

This work is organized as follows, in Chapter 2 we quickly summarize the main concepts concerning RDF and the characteristics that make it suitable for representing information on the World Wide Web. Then there will be a brief presentation of how keyword search has been addressed in the literature. In the same chapter there is also a summary of the basic operation of TSA+BM25 and TSA+VDP. We focus on the evaluation metrics first introduced in the original paper. In Chapter 3 you can find the details of how we could reproduce the original TSA+BM25 and TSA+VDP systems correctly. A final table also shows that our results are in line with the original results. In addition, there are comments on how the code can be optimized to make the understanding of the code easier. The two methods of update management are described in detail in Chapter 4. In Chapter 5 there is a close analysis of the results obtained from the two systems, which are compared against each other and in relation to the original system. Chapter 6 contains the conclusions and some final considerations.

2 Background

2.1 RDF

RDF, or *Resource Description Framework*[1], was introduced by the W3C organization as a graph model standard for representing information on the Web, more specifically how resources or entities are defined and interact. This standard was developed to ease the exchange of information between different applications and to interlink data on the Web.

The basic idea behind RDF is to put in relationships two resources and then expand it to other entities. In RDF the relationship between two entities is expressed as a statement in the form of a triple: <*subject*> <*predicate*> <*object*>. The two entities are <*subject*> and <*object*>, while the <*predicate*> identifies the property of the relationship and the unique direction of reading. In fact, the RDF graph is considered a directed labeled graph, where the set of nodes is the set of subjects and objects; the set of arcs is the set of the couples (subject, object). There can be three types of data in RDF statements or triples: IRIs, literals, and blank nodes. An International Resource Identifier (IRI) is the unique, global identifier of a resource, in fact one of its subsets is the URL that is commonly used to surf the World Wide Web. A literal is simply a value like strings, names, and numbers usually each literal is annotated with a precise datatype to aid interpretation of the data. Finally, blank nodes are resources without a specific IRI, used when a unique identifier is not required. With an IRI we can identify each component of a triple, a subject, an object or a predicate. A literal instead can only be an object. A blank node can serve as a subject or an object.

In the example of Figure 2.1, <`Bob`> is the subject, <`is interested in`> is the predicate and <`The Mona Lisa`> is the object which in turn becomes the subject if we look at the following triple <*the Mona Lisa*> <*was painted by*> <*Leonardo Da*

[1] `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`

Vinci>. The first statement describes the fact that Bob is interested in the Mona Lisa portrait. In DBpedia, a big knowledge graph, the Mona Lisa painting is identified as `https://dbpedia.org/page/Mona_Lisa`

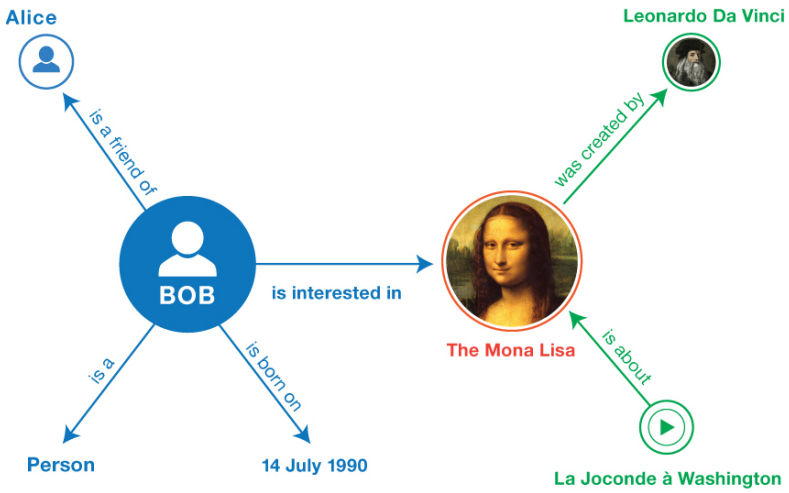

Figure 2.1: Example of a dummy RDF graph provided by W3C community

2.1.1 SPARQL

SPARQL[2] is another standard developed by W3C to query, extract and manipulate RDF graphs. A SPARQL query is defined by a set of triples that compose a pattern to be found, or matched, on the RDF graph. In the set of triples defining the pattern some of the resources might be unknown and substituted by aliases. This pattern can be simple or complex; SPARQL provides advanced tools to let the user look for pattern aggregations, use subqueries or grouping and sort the results. The output of the query can vary based on the query type defined before performing the search. A result can be a boolean value if the query type is ASK, a complete RDF subgraph if it is used the CONSTRUCT type, or with the most common SELECT, the results are composed only by the matched data triples

2.2 Keyword search on structured data

In general, we can define a keyword search problem as follows:

[2] `https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`

Problem 1 *Given a keyword query K and an RDF graph G, find an answer S for K such that the subgraph G_A found has the set of matched keywords $K_{G_A} \subseteq K$ without useless nodes or connections. This set induces another set, the matched triples that form the answer[14].*

In the literature, there exist two main approaches, one is where the keyword query is translated into a SQL/SPARQL queries. It is usually a **schema-based** method because it uses the knowledge of the schema of the dataset to formulate a set of SQL/SPARQL queries that can answer the information need of the user. The other method is **graph-based**, with traversal algorithms the graphs is explored to find directly the answers.

For the relational dataset there are up to now two main paths, Candidate Networks (CN) and Steiner Trees. One of the first papers to present CN were DBXplorer[1] and DISCOVER [13], there the database is seen as many interconnected tuples, then some networks centered in the query keywords are proposed as solution. On the other hand BANKS system [3] use the concept of Steiner Trees, where at the leaf node we find the keywords, so it considers the RDB as a graph, then the algorithm produces top-k results where the answers are connected subgraph. The Steiner approach and top-k answers open some criticality to large graphs(it is a NP-problem). Studies like [4] developed a dynamic programming algorithm to directly solve the Steiner Tree problem, successful only with a single keyword. QUEST[2] is a recent system where using heuristic rules, machine learning techniques and a Steiner tree they produce an SQL query. A recent article [27] tries to overcome the limitation of Steiner trees using parallel computing managing to fully utilize CPU and even GPU to speed up the computation of the top-k answers graphs introducing the concept of Central Graphs applied specifically to knowledge graphs. In RDB, a similar thing was developed by [28] where using Map-Reduce the keyword search problem was split in a distributed environment and performed in parallel. In [19] the authors propose a system to help the user writing the next terms of the SQL query automatically. The information extracted from the schema can be used to understand the roles of the keywords, in [17] they try to grasp the user intent, then rank join trees that cover the found keyword roles. Again considering schema-based systems, we find SPARK[21] a keyword search system for relational databases where keyword terms are enriched by Wordnet synonyms.

Another strategy to query RDB is [22], MRF-KS uses the virtual document idea. The graph structure is derived from the relational database. The system

then extracts support graphs and metadata. The nodes and literals are converted into text documents, pruned and ranked when the user query is considered.

QUOIW[11] is a system that let the user explore both worlds: SQL and SPARQL. They design some heuristic lower bounds and a bipartite graph matching-based best-first search algorithm to build step-by-step the correct query via entity and predicate blocks. There are some interesting studies for RDF. Specifically, [29] propose a similar thing to the previous system, QUICK translate keyword-based queries in SPARQL queries where the user select the next ranked "blocks" proposed, again helped by the RDF schema. Then the prevalent approaches build on indexing techniques and search algorithms to find substructures/sub-graphs that connect the data elements matching the keywords. Graph summarization was the method used in [24], [18], [31], [20] to produce SPARQL queries without working on the entire RDF graph.

The virtual document idea of MRF-KS can be applied to graph data. RDF's most relevant keyword search system are SUMM[25] and SLM[9], systems that exploit the virtual document idea to process the RDF graph to retrieve the query answers. The former partitions with BFS the graph into blocks connected by single nodes. In the phase where the query comes into play the subgraphs are merged and pruned to produce a top-k ranking based on the weight of the tree produced. The latter instead proposes to index each document created for every single triple of the RDF graph, the documents are then indexed and retrieved. The proposed ranking is based not only on the similarity of the text with the query but also the structure of the subgraphs is taken into account. The virtual document approach was found to be promising when exploring huge RDF graphs.

Indexing the single triple is also the concept by [16], they created a web application where, the query can be submitted by a REST API. The results of a keyword query are presented in multiple ways. There they used Elasticsearch to index and retrieve all the relevant triples.

Recently other methods were developed under the graph-based approach: [14] used KMV synopses to compute set similarity during a single pass of the RDF graph. Those allow the extraction of properties domain and class instances. The final process is a precise SPARQL query. The semantic-driven approach developed by [10] focuses on the semantic aspect to improve the final quality of the answers by automatically finding subgraphs, with respect to the traditional retrieving model. In [30] it is introduced KBCS system that is focused on the

RDF structure, here the graph is converted into a more simple and small graph. Then a community algorithm find subgraphs. Finally a mapping of the query to the index create a set of ranked tree answers based on compactness.

A promising idea is presented by [23] proposes to transport the problem into a vector space. Embeddings are computed for the graph entities and queries, then the query is compared to the graph embeddings to retrieve the most similar entities.

In this scenario, we could not find any work that considered and developed ideas regarding updates in a keyword search system on RDF, so trying to explore this aspect will be the core and the main contribution of this work.

2.3 TSA - Topological Syntactical Aggregator

The work present on this dissertation is entirely based on *Topological Syntactical Aggregator* that is the name of the new keyword search system developed in [6], [7], [8], [5]. Dosso and Silvello proposed a new way to search in RDF graphs using the virtual document technique. This system falls into the graph-based systems to retrieve data because it does not use any knowledge about the graph schema. They even developed a new evaluation framework and introduced a new metric to evaluate the system, because such framework did not exist. The system follows the best-match paradigm: the user keyword query is answered returning a ranked list of subgraphs. Opposed to the exact match approach where after submitting a specific and unambiguous SQL query the result is only one and precise. The system is divided into two parts to fulfill the task: offline and online.

Figure 2.2 is taken directly from that paper it illustrates a running example and all the different steps to reach the goal. In the offline part, the system performs clustering on the RDF graph, in this way similar concepts will be clustered together while dissimilar objects will be separated. For example, a subgraph could focus on a specific film and its characteristics. The result is a set of unique clusters called representative subgraphs. The operation is supported by some metadata such as the in/out-degree of the nodes. All the subgraphs are then converted into text documents by considering all their IRIs and literals. In the online part the keyword query enters the game, in the figure we can see that the text documents are retrieved by exploiting cutting-edge IR techniques producing the first ranking. The documents are then transformed back into

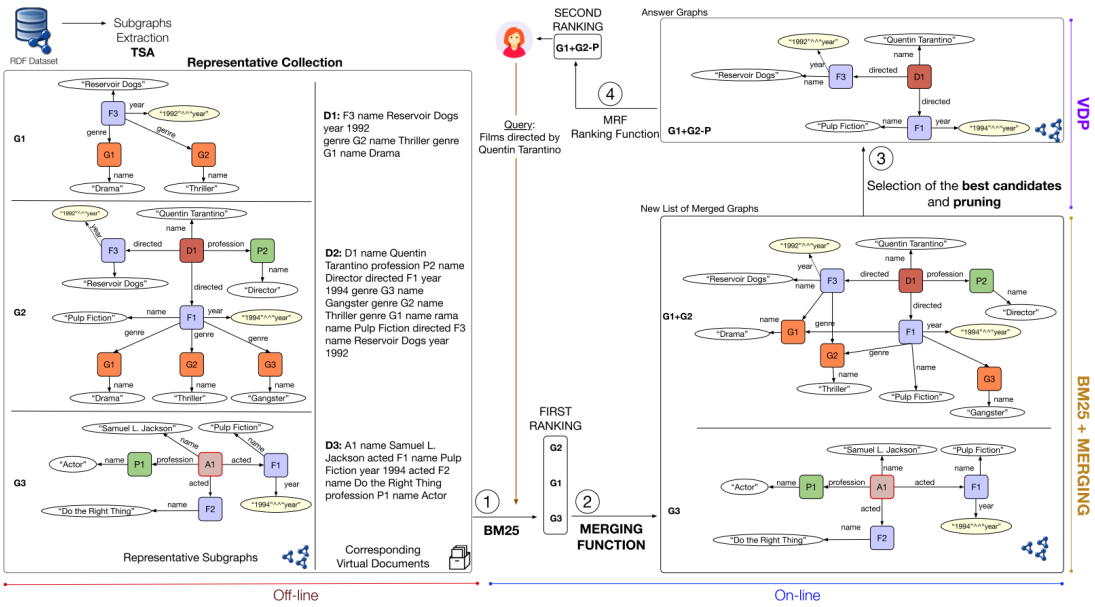

Figure 2.2: Running example of the TSA system, both BM25 and VDP pipelines are shown

RDF graphs where with pruning and merging operations the final raking is presented to the user, or can be refined to improve the quality of the response.

2.3.1 Offline phase

Algorithm 1 presents the steps to build representative subgraphs, the main part of the offline phase. Before explain in detail we have to give some definitions:
In a RDF graph $\mathcal{G}$, v is a node;
$\delta^-(v)$ is the out degree of a node v that is the number of out going links, that is the cardinality of its neighbors $\mathcal{N}^-(v)$;
$\delta^+(v)$ is the in degree of a node v that is the number of in going links;
The source set S is composed by all $v \in \mathcal{G}$ with $\delta^-(v) > \lambda_{out}$, whereas the terminal set T is all $v \in \mathcal{G}$ with $\delta^+(v) > \lambda_{in}$, with λ_{out} and λ_{in} two integer thresholds values. There is also τ which is the radius of the final subgraph. $\mathcal{L}$ is the list of the top-k frequent predicates in $\mathcal{G}$

The algorithm starts considering a source node s as a starting point for a subgraph, then performing a BFS approach it explores the neighborhood of s adding all its literals. If terminal nodes are present, together with their liters nodes, are inserted into the subgraph. Finally, the exploration moves to other source nodes if the predicates are in the list $\mathcal{L}$ and if the source nodes are

there within a radius τ. Every time a source node is explored it becomes black and it will never be considered again; it easy to spot those terminal nodes can be present in many subgraphs whereas a source node only in one. The next iteration starts from the next available white source node.

By tweaking the various user-accessible parameters, it is possible to change the final result. Increasing τ will result in bigger and fewer subgraphs causing more source nodes to be clustered together. We can have fewer subgraphs if λ_{out} is too high because fewer source nodes will satisfy the constraint. If λ_{in} is big then fewer terminal nodes will be considered resulting in smaller subgraphs. Each of the datasets considered to test the system had a different set of parameters, with the exception of τ which is linked to the specific query structure.

2.3.2 Online phase

For the online part, they propose two different pipelines, BM25 and VDP, the latter is always executed after BM25. The first is called BM25 because it is based on the famous retrieval model and BM25 function to weigh the documents given a query. The online part starts by considering the user keyword query creating a ranking of the text documents. From this raking only the first 10000 positions are considered and pass to the next process where the text documents are discarded in favor of the corresponding RDF graphs, as we see in Figure 2.2. Those graphs go through a merging function. There given a threshold the subgraphs are compared and merged if the overlap is greater than the threshold(threshold seen with respect to the smaller one). This new graph is then put in comparison to the next graphs. The comparison proceeds for the first l graphs of the raking until the size of the new merged graph does not reach 1000 or l is fully explored. The new graphs are called Merged Graphs. Those merged graphs undergo a second ranking ending BM25 pipeline.

The second pipeline starts from those ranked graphs considering only the top n of that ranking. The graphs are considered as a whole in a set-based union. From every source node present VDP starts a BFS with radius τ generating new subgraphs, but those that do not contain all the keywords of the query are discarded. In the figure those are the "best candidates". To remove any noise present in the "best candidates", i.e., unnecessary triples that would lower the final score, cleaning is performed starting with the triples farthest from the center of the subgraph.

The final ranking submitted to the user is based on a variation of Markov Random Fields described in [22] that consider unigrams and bigrams. A unigram is the single term of a sequence like *The sky is blue*, *blue* is a unigram; a bigram is composed by two adjacent tokens, such as *is blue*. This part is full of functions and definitions, which we briefly report and are crucial for later understanding of the results. The TSA paper also presents a detailed section in which they focus on system evaluation. In fact, the researchers have defined a new metric, tb-DCG. Which incorporates and recalls the concepts of the better-known nDCG, famous in the IR domain.

The score assigned to a graph is as follows where g is the graph, and Q is the query. The score is the sum of two distinct sums one concerning the unigrams f_U the other f_B the bigrams, each term is considered within the specific graph g.

$$score(g, Q) = \sum_{q_i \in Q} f_U(q_i, g) \quad + \quad \sum_{\{q_i, q_{i+1} \in Q\}} f_B(q_i, q_{i+1}, g)$$

In the function f_U we find g^* that is the text document associated to graph g and C is the overall text collection; α_U is the Dirichlet's smoothing factor that considers the average of the length of the document.

$$f_U(q_i, g) = ln[(1 - \alpha_U)P(q_i|g^*) + \alpha_U P(q_i|C)]$$

The two probabilities are:

$$P(q_i|g^*) = \frac{wtf(q_i, g^*)}{\sum_{d \in C} wtf(q_i, d^*)} \qquad P(q_i|C) = \frac{\sum_{u \in C} tf(q_i, u^*)}{\sum_{d \in C, w \in u^*} tf(w, u^*)}$$

with wtf we have the weighted term frequency that rates the frequency and the position of the node in the graph with respect to the center whereas tf is the simple term frequency.

$$wtf(q_i, g^*) = \sum_{v \in g} e^{\frac{-(w_s(g^*, v) - w_s(g^*, s_g))}{2\sigma^2}} ft(q_i, v^*)$$

Here v is a node in g, s_g is the source node of the graph, and σ is the kernel control spread, set to 1. The term $w_s(g^*, v)$ is the relative static weight of the node v in g, defined as the path with the minimum weight between all paths from the origin to v:

$$w_s(g^*, v) = min_{p_{s_g \to v \in g}} w_s(p_{s_g \to v})$$

The static weight is instead defined as follows:

$$w_s(p_{s_g \to v}) = \sum_{e \in p} we_s(e) + \sum_{x \in p} wn_s(x)$$

where the edge weight of edge e $we_s(e)$ is set to 1; the static weight of the node x is:

$$wn_s(x) = \frac{1}{ln(e + \delta^-(u))}$$

2.3.3 Evaluation

The evaluation is a core component to determine the quality of the answers produced by a system, in IR the most used framework is the Cranfield one, where the documents in the pool are immutable, and the ground truth is hand-made by the assessor how decides if a document is relevant for a specific topic. In the keyword search over structured data things are a bit different because documents are not fixed and can widely vary by intervening on various parameters. The ground truth G_{t_k} is constructed by transforming the user keyword query into a construct SPARQL query to have in output a graph; then all the triples extracted can be considered relevant.

The final ranking submitted to the user is characterized by the fact that at the top are the most relevant documents with a higher ratio of useful to useless triples, so we can say it is "top-heavy". The aim is to have documents down in the rankings that do not have triples already contained in documents ranked higher, penalizing those that have.

To reach the objective they defined some metrics that will be used later in the experiment chapter[6].

The Signal-to-noise ratio given a topic t_k, the corresponding ground truth G_{t_k}, the set S of all the relevant triples in $G_i \in R_k$ with $j \in [1, i[$ is:

$$SNR(G_i) = \frac{|(G_i \cap G_{t_k}) \setminus S|}{|G_i|}$$

Suppose the SNR is greater than a value λ, set between zero and one. In that case, the graph is considered relevant because it represents the triples returned for the first time, at the numerator, with respect to all the triples of the graph measured at that moment. We will encounter other important metrics later,

such as recall, precision, precision@c and the new tb-DCG. The recall is a value that does not express the quality of the ranking but can give an idea about the number of relevant triples found by the system and its effectiveness. On the contrary, precision and precision@c tells us how pertinent many triples are present in the ranking because it represents the ratio between relevant triple and all the distinct triples returned by systems: precision can consider the noise present in the final ranking. The value c is the level at which we stop evaluating the precision, most of the time the bar is set to 1, 5 and 10, in other words the precision at the first, five or ten results. A high precision means that the system can rank the graphs in an optimal way.

$$recall(R_k) = \frac{\bigcup_{G_{i \in R_k | SNR(G_i) \geqslant \lambda}} (G_i \cap G_{t_k})}{|G_i|}$$

$$precision(R_k) = \frac{\bigcup_{G_{i \in R_k | SNR(G_i) \geqslant \lambda}} (G_i \cap G_{t_k})}{|\bigcup_{G_{i \in R_k}} G_i|}$$

$$prec@c(R_k) = \frac{|\bigcup_{G_{i \in R_k \ | \ SNR(G_i) \geqslant \lambda \ \wedge \ i \in [1,c]}} (G_i \cap G_{t_k})|}{|\bigcup_{G_{i \in R_k \ | \ i \in [1,c]}} G_i|}$$

In the IR domain, there is the Discounted Cumulative Gain [15] a measure mostly used for evaluating web search engines with the hypothesis that highly relevant documents are more useful to the user that will click them versus others in the lower positions. Creating a gain that is cumulated at every position till a point in which it starts to be discounted, each document does not bring the full usefulness to the user. In [6] it is defined a new metric for the keyword search in RDF filed, the *triple based-DCG*.

$$tb\text{-}DCG_b(R_k) = \sum_{i=1}^{n} RG_B(G_i) \quad RG_b(G_i) = \begin{cases} GRW(G_i) & \text{if } i \leq b, SNR(G_i) > \lambda \\ \dfrac{GRW(G_i)}{log_b i} & \text{if } i > b, SNR(G_i) > \lambda \\ 0 & \text{if } SNR(G_i) \leq \lambda \end{cases}$$

Each term $RG_b(G_i)$ in the summation is the Relevance Gain of graph G_i, and depending on the position of the graph its relevance change, with b set to 2 every term beyond that position will be discounted by a specific factor. The value λ that we recall being the relevance parameter, sets the threshold at which a graph

stops being relevant. GRW is the last piece of this complex evaluation system. It is the weighting function, similar to the SNR. Still, here at the denominator, we have the cardinality of the ground truth, so it can be seen as a sort of attendance to the truth, where graphs with unique, and first-time seen, triples are weighted more heavily.

$$GRW(G_i) = \frac{|(G_i \cap G_{t_k}) \setminus S|}{G_{t_k}}$$

Algorithm 1 TSA

Input: RDF graph $\mathcal{G}$, source set S, terminal set T, list $\mathcal{L}$, integer radius τ
Output: list $sbgr$ of representative subgraphs

```
1: sbgr ← ∅
2: Q ← ∅
3: for each s ∈ S do
4:    if s.color = white then
5:       Q.enqueue(s)
6:       s.τ ← τ
7:       G ← new Graph()
8:       while Q ≠ ∅ do
9:          v ← Q.dequeue()
10:         v.color ← black
11:         radius ← v.τ − 1
12:         for each u ∈ N⁻(s) do
13:            if u ∉ S ∧ u ∉ T then
14:               //Accessory node
15:               G.addTriple((s,u))
16:            end if
17:            if u ∈ T then
18:               // Terminal node
19:               G.addTriple((s,u))
20:               for each w ∈ N⁻(u) do
21:                  if w.isLiteral() ∨ (w ∉ S ∧ w ∉ T) then
22:                     G.addTriple((u,w))
23:                  end if
24:               end for
25:            end if
26:            if (p(s,u) ∈ L) ∧ (radius > 0) ∧ (u.color ≠ black) then
27:               if u ∈ S ∧ u ∉ T then
28:                  u.τ ← radius
29:                  Q.enqueue(u)
30:                  G.addTriple((v,u))
31:               end if
32:            end if
33:         end for
34:      end while
35:      sbgr.add(G)
36:   end if
37: end for
38: return sbgr
```

3

The Replicability work

In this chapter, I explain the methodology followed to reproduce and thoroughly understand the TSA system, the first of the two objectives of this thesis, to develop later the update management concept that will be discussed in more detail in the following chapter. Section 3.1 describes all the software and data used to learn and test the system; Section 3.2 covers the logic path followed to reach the objective. Finally Section 3.3 contains a quick summary of the tests conducted tests to verify the system's operability. Most of the work done in this part was a sort of reverse engineering job.

3.1 Software and data

Here we present the software that was and still is being used in the study.

- PostgreSQL (version 14.0)[1]
- Blazegraph[2]

PostgreSQL is currently one of the most famous, reliable and flexible open-source database; Blazegraph, instead is a native high-performance graph database to surf and perform SPARQL queries. Unfortunately, the documentation of Blazegraph is poor and the only support you can find is limited to the GitHub page within the small community. Because of that the learning curve has been steep, but it gave me insight into a precious and powerful application.

3.1.1 Data processing

I started my journey into the system developed by Dosso and Silvello [6] by downloading the dataset they used: the synthetic database LUBM (Lehigh

[1] https://www.postgresql.org/
[2] https://github.com/blazegraph/database/wiki/Main_Page

University Benchmark)[3], LinkedMDB[4] and IMDB[5].

LUBM is a synthetic database composed of fake and automatically generated data based on user requests and following a pattern decided at an earlier stage based on a well-defined ontology concerning the university domain, as can be appreciated in the figure. The database is used as a test to evaluate and refine Semantic Web repositories in a standard way. LUBM also comes with a set of 14 queries. The dataset is generated by choosing the number of universities. If we set just one university, the result is 130 thousand triples. We created a dataset of 1 million triples considering ten universities.

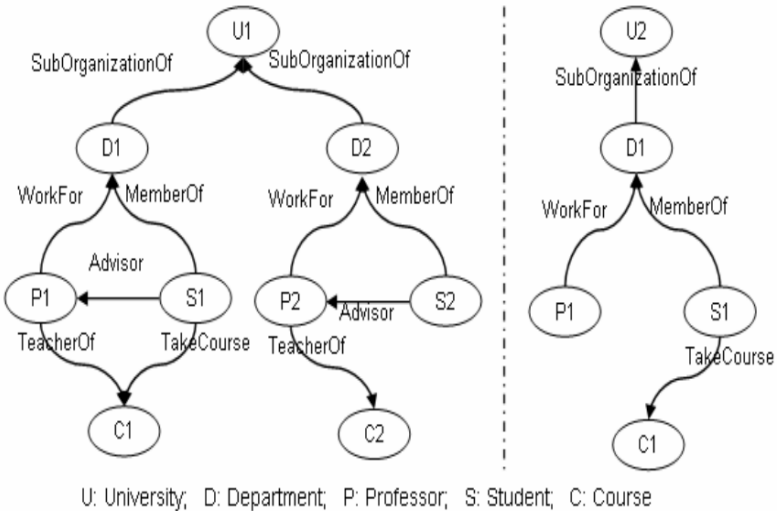

Figure 3.1: Original schema upon which the LUBM data are generated. there are two distinct schemes

LinkedMDB and IMDB are datasets concerning film and movies and all the entities that revolve around them, below there is an example. LinkedMDB [12] is a native RDF dataset part of the expanding universe of the Linked Open Data concept, it also contains some links to the IMDB dataset. This RDF graph is composed of 6.1 million triples. IMDB, the biggest movie database, is shared as a relational dataset, so we start from this to illustrate the brief data processing we performed, below there is the ER schema[26]. First of all, we used only a subset of the data that IMDB of what is made available:

- `title.basic` contains the feature of a product, like the primary title, the language, genre etc...

[3] `http://swat.cse.lehigh.edu/projects/lubm/`
[4] `https://www.cs.toronto.edu/~oktie/linkedmdb/`
[5] `https://www.imdb.com/interfaces/`

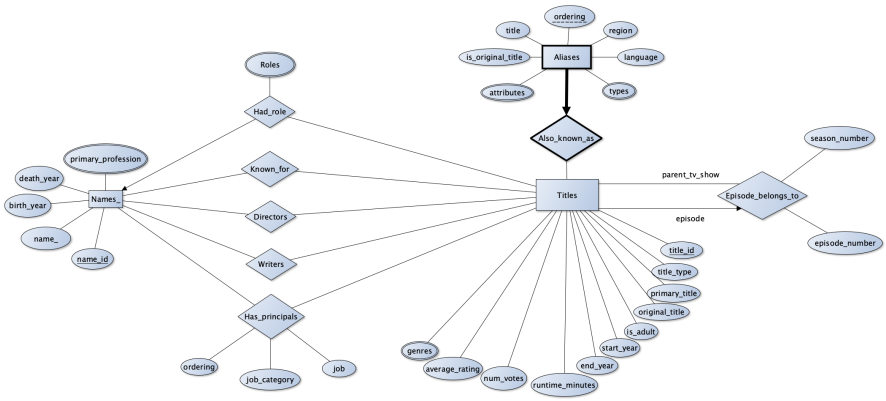

Figure 3.2: ER Schema of the IMDB dataset

- `name.basics` include the name, birth and death year and the principal works a person is known for.
- `title.principals` contains cast/crew for titles.
- `title.crew` has director and writer of the title.

These two datasets came with a list of 50 construct queries each, half for the training and the other for testing.

The relational database was converted into an RDF graph seeing the column names as predicates, except for the first one, always the primary or foreign key, that acted as the subject. The objects were the content of the adjacent cells. As of today, November 5, 2022, considering only those files the RDF database counts more than 300 million triples. Because of the size, we developed a Java class to reduce the size without losing the original structure. we sampled some subjects in a random way from which to run a BFS, with a radius equal to 3, leading to a final size of one-tenth the original size obtaining a more manageable dataset.

3.2 The recipe

The original system was developed entirely in Java, and it is available at the following URL: `https://bitbucket.org/account/user/keywordsearchrdfproject/projects/TSAC`.

The first thing to notice is how it is very well structured, there are 6 modules: `efficient backtracking`, `rdf_blanco`, `yosi_rdf`, `terrier_custom`, `rum` and `offline_rdf_clustering`. The latter is the core of the whole TSA+BM25 and TSA+VDP systems and the one that we studied to present this

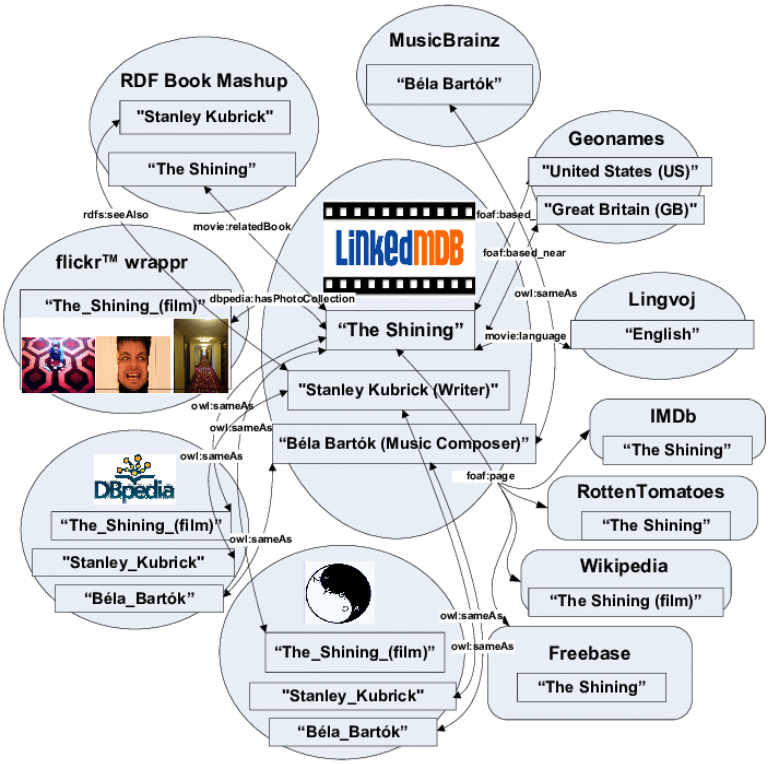

Figure 3.3: Sample of a LinkedMDB entities

research. The other consist in the rewrite of other RDF search systems: SLM[9] SUMM[25] and MRF-KS[22].

The import of those modules into the IDE was easy and the wizard process was straightforward. The first thing done was to understand how the different modules were interconnected; in particular `offline_rdf_clustering` extensively uses many of the classes in `rum`, which stands for RDF Useful Methods. In `rum` we find most of the classes that initialize the interface with the SQL database or the RDF one. In this case there is a specific class that eases the import of the RDF data(in the format of TURTLE or NTRIPLES) inside a Blazegraph RDF repository; we also have classes, that define all the objects created from scratch used in the project, such as the notion of a *triple*. Finally, there is everything necessary for the generation of the final assessment of a complete run. The abundance of code and classes proved to be a double-edged sword; many times we found myself going through the tree of exceptions to identify the real cause of the error, sometimes it was a minor oversight, and other times it gave me an insight of how the system structure was set.

In Figure 3.4 we have summarized the most important processes executed by

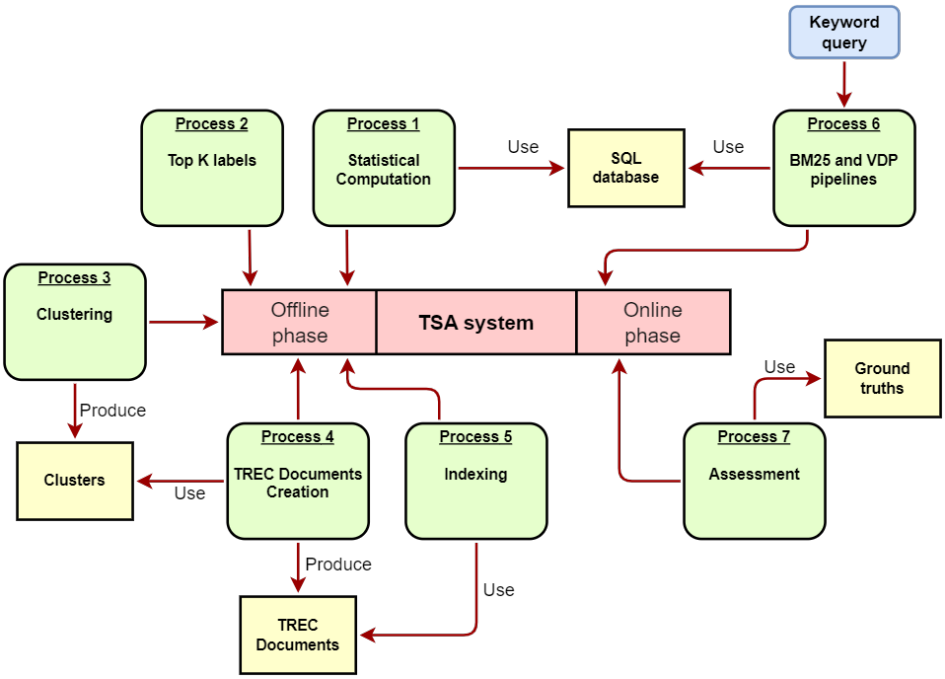

Figure 3.4: Schema of the TSA processes and elements the interact with

TSA system, arranged chronologically. Those are divided into two categories, the ones for the *offline* part and others for the *online* one; as already explained in detail in the previous chapter. We started the analysis of the code from *Process 1: Statistical computation*. Here the program analyzes the SQL dataset to set all the nodes and labels metadata. In this phase we have figured out how to build the SQL tables to simulate and RDF environment, specifically with the *triple store* table that recall the RDF concept of the **s**, **p**, **o** structure where we have a column for the subject, one for the predicate and a last one for the object. The program makes use of two other auxiliary tables: one for the labels $\mathcal{L}$, the set in which the predicates label are put together with their frequency; and one for the nodes, a table that stores the statistical information required to perform the TSA algorithm, the in and out degree for each node. Below are the codes used to make the tables. The SQL database is later used by the VDP pipeline to explore the neighborhood of a node in the search of all the query terms.

```
CREATE TABLE triple_store (id_ SERIAL,
subject_ VARCHAR(600), predicate_ VARCHAR(600), object_ VARCHAR(600))
```

Code 3.1: SQL command create the triple store

Then we created one index for the *subject* column and another for the *id*. As

explained in the previous chapter, those are the most accessed fields during the computation. All of this allowed us to cut down on execution time drastically.

```
CREATE TABLE label (label_name VARCHAR(600), frequency INTEGER)
```

Code 3.2: SQL command create label table

```
CREATE TABLE node (id_ SERIAL, node_name VARCHAR(600), in_degree REAL
    , out_degree REAL, iri_out_degree REAL)
```

Code 3.3: SQL command create the node table

The second thing to consider and that was under my direct control was the ground truths production, inside Blazegraph we needed to perform all the SPARQL queries to retrieve all the triples that answer correctly to a query. In an Information Retrieval domain those are considered the relevant triples. Blazegraph is not included in the system figure because it was used almost behind the curtain to get the system ready to start. Given the fact that the system returns RDF graphs as answers, the queries where in the `construct` type. The queries were shared without the prefixes, single words that act as alias shortening the URIs; those are necessary to run the SPARQL queries, so we looked into the RDF graph performing some generic and exploratory queries to find the correct ones.

Here we report an example: the query asks for some characteristic of the works of a director, for instance Michael Bay.

```
prefix imdb: <https://www.imdb.com/>
prefix imdb_primary_profession: <https://www.imdb.com/primary_profession/>
prefix imdb_title_type: <https://www.imdb.com/title_type/>
prefix imdb_genres: <https://www.imdb.com/genres/>

CONSTRUCT WHERE {

                ?actor_id imdb:primary_profession imdb_primary_profession:director ;
                imdb:primary_name "Michael Bay" ;
                imdb:known_for_titles ?title_id .
                ?title_id imdb:primary_title ?title ;
                imdb:title_type ?type ;
                imdb:genres imdb_genres:adventure .

                }
```

Figure 3.5: Query example: here the information need is to find all Michael Bay's works, their type, title and genre

Blazegraph works with one repository at a time, so to manipulate the three datasets we had to work with them separately. After uploading the RDF graph into Blazegraph, in `rum` we found a class to read the triples from the repository and to insert them inside the RDB. Unfortunately, the method had some design

```
<https://www.imdb.com/name/nm0000881> <https://www.imdb.com/primary_profession>
                                          <https://www.imdb.com/primary_profession/director> ;
                                      <https://www.imdb.com/primary_name> "Michael Bay" ;
                                      <https://www.imdb.com/known_for_titles>
                                          <https://www.imdb.com/title/tt0117500> ,
                                          <https://www.imdb.com/title/tt0120591> ,

<https://www.imdb.com/title/tt0117500> <https://www.imdb.com/primary_title> "The Rock" ;
          <https://www.imdb.com/title_type> <https://www.imdb.com/title_type/movie> ;
          <https://www.imdb.com/genres> <https://www.imdb.com/genres/adventure> .

<https://www.imdb.com/title/tt0120591> <https://www.imdb.com/primary_title> "Armageddon" ;
          <https://www.imdb.com/title_type> <https://www.imdb.com/title_type/movie> ;
          <https://www.imdb.com/genres> <https://www.imdb.com/genres/adventure> .
```

Figure 3.6: The answer (partial) to the query in Fig. 3.5, here the results is reported as an RDF graph in TURTLE format

choices that were not optimal: the entire RDF graph was loaded into the main memory. As a consequence during the import of IMDB the system crashed. To give you an idea, the Blazegraph repository of LinkedMDB was 1.3 gigabytes, and the IMDB one was 32.3 gigabytes. So we made some refinements in a way to explore the graph incrementally. This workaround carried out the load into PostgreSQL in a matter of minutes.

The third process is one of the core processes, together with the sixth, it generates the clusters, the subgraph around a specific topic, following the TSA approach. Here some parameters could be tweaked, for instance the threshold for in and out-degree to discern between source and terminal nodes. Processes 4 and 5 are autonomous and revolve around the TREC documents produced from the cluster and than indexed using a Terrier implementation. Process 6 is in charge of all the online parts of the TSA system, in there we have both pipelines, BM25 and VDP. VDP because of its nature, explores the graph and so makes many calls to the RDB, that we have stated before recreates and fully replaces the RDF graph by Blazegraph. The keyword search assessment is done inside process 7; here, there is a comparison between the retrieved query answers and the ground truths. All the ground truths needed to be saved in one folder ordered from 1 to n.

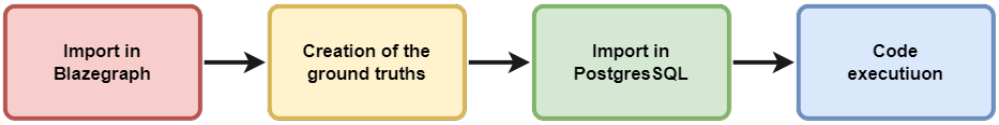

Figure 3.7: Logic schema followed to reproduce the report data

The study of the code resulted in me identifying the flowchart in Figure 3.7 to fully execute the system and make a run for a dataset. Given the fact that LUBM

dataset can be both a small dataset and a large one without any additional work, we used it in two different sizes, 1 and 10 million, it was my choice, along with LinkedMDB to conduct some of the final tests done before we could claim full mastery of the system and try to process IMDB.

3.3 Testing

In this section we report my final test results comparing them with the one from [6].

Dataset	percentage of update	Systems	tb-DCG	recall	prec@1	prec@5
LUBM 1 M	BM25	My run	0.292	0.361	0.112	0.120
		Original	0.284±0.06	0.344±0.07	0.120±0.04	0.131±0.05
	VDP	My run	0.260	0.101	0.225	0.299
		Original	0.243±0.05	0.091±0.04	0.224±0.06	0.306±0.05
LUBM 10 M	BM25	My run	0.288	0.489	0.077	0.110
		Original	0.281±0.07	0.505±0.07	0.082±0.04	0.111±0.05
	VDP	My run	0.350	0.387	0.021	0.021
		Original	0.343±0.06	0.394±0.06	0.025±0.06	0.022±0.06
LinkedMDB	BM25	My run	0.183	0.855	0.000	0.006
		Original	0.171±0.01	0.916±0.02	0.060±0.00	0.003±0.06
	VDP	My run	0.537	0.861	0.034	0.032
		Original	0.429±0.04	0.458±0.08	0.037±0.00	0.036±0.00
IMDB	BM25	My run	0.152	0.317	0.121	0.061
		Original	0.067±0.01	0.273±0.36	0.011±0.00	0.009±0.00
	VDP	My run	0.255	0.281	0.031	0.016
		Original	0.308±0.04	0.363±0.06	0.006±0.00	0.006±0.00

Table 3.1: Results obtained in my attempt to replicate the TSA+BM25 and TSA+VDP systems

Dataset	Systems	tb-DCG		recall		prec@1		prec@5	
LUBM 1 M	BM25	0.008	(+2.8%)	0.017	(+4.9%)	-0.008	(-6.7%)	-0.011	(-8.4%)
	VDP	0.017	(+7.0%)	0.010	(+11.0%)	0.001	(+0.4%)	-0.007	(-2.3%)
LUBM 10 M	BM25	0.007	(+2.5%)	-0.016	(-3.2%)	-0.005	(-6.1%)	-0.001	(+0.9%)
	VDP	0.007	(+2.0%)	-0.007	(-1.8%)	-0.004	(-16.0%)	-0.001	(-4.5%)
LinkedMDB	BM25	0.012	(+7.0%)	-0.061	(-6.7%)	-0.060	(-100%)	0.003	(+100%)
	VDP	0.108	(+25.2%)	0.403	(+88.0%)	-0.003	(-8.1%)	-0.004	(-11.1%)
IMDB	BM25	0.085	(+126.9%)	0.044	(+16.1%)	0.110	(+1000%)	0.052	(+577.8%)
	VDP	-0.053	(-17.2%)	-0.082	(-22.6%)	0.025	(416,7%)	0.010	(+166,7%)

Table 3.2: Raw and percentage differences in my replicability work results.

We can observe from Table 3.1 and Table 3.2 that at the end of the multiple tests, in almost all metrics, we was able to improve sometimes on the original

results by playing with the available parameters such as the number of graphs considered by BM25 or VDP at the beginning of their execution or the percentage of overlapping. The results on the LUBM synthetic dataset are closer to the original ones in all the metrics, in both the dimension tested, 1 million and 10 million triples. The difference ranges from BM25 and VDP performed a little better on tb-DCG. In particular, for VDP we raised 250 to 400 the number of graphs considered before starting the pruning phase. As we can see, the tb-DCG and recall values went up; even precision@1 was marginally better, but this falls within the margin of error.

For LinkedMDB, the gain obtained for the tb-DCG metric was not repeated for recall which had slightly lower results. In this case, we modified the maximum number of graphs BM25 tries to analyze for the merging. In my test, this value was 15, while in the baseline was only 5. The cutoff for the overlapping remained fixed at 20, we even tried to change it to different values like 15 or 12 but the final results were always similar to the ones reported in the table. With this dataset, we can see that BM25 in the precision cases did not perform well. In particular in the prec@1 case it never returned a relevant answer. VDP, on the other hand, had the most significant improvements, especially in the recall. For LMDB and IMDB we decided to increase the number of graphs considered by VDP for query-graph creation: from 100 we raised it to 120. As a result, the first 120 BM25 ranking graphs were merged into one large graph from which VDP then proceeded with the center selection phase and the elimination of unnecessary and noisy triples. VDP then worked by having many more triples available. So with this little change, VDP retrieved many more relevant triples and ranked them better than the baseline. The relevant graphs retrieved were even higher than my BM25 results. Finally, we can say that VDP in LinkedMDB obtained good results improving in every major aspect, in the precision values the results obtained were compared with the baseline ones.

Separate discussion for IMDB. Since it could not be reconstructed faithfully to the original and having been reduced for practical matters, this dataset has generated great results nevertheless. Compared to the baseline BM25 in all the cases had higher results, while VDP got a bit lower results, especially in tb-DCG and recall. In contrast, the precision values are similar to those in the original paper. Those results may be poisoned by the fact that the overall noise, composed by irrelevant triples is less than in the older dataset: the work of BM25 is simplified, and therefore it can retrieve answers in a better way. As already

observed in [6] BM25 retrieves more graphs, but in my run, VDP cannot fully take advantage of that.

These results suggest that we could achieve the objective and reproduce the system. Many of the results are within 1 to 8% with respect to the original paper, but in some cases, the increment is up to 50% in the LMDB dataset or more than 126% if we consider BM25 in the IMDB dataset. The original paper also included some tables with the offline execution time, but those were left behind: the systems have proven to be significantly faster on my machine.

4

Update idea

In this chapter, we will describe the main idea behind this work and how it was developed. An update on a graph is the addition, modification, or removal of one or more triples that make up that graph. The goal is to add, remove or modify information. After noticing that the issue of updates on an RDF system had not yet been considered, we thought we might modify the already developed system in [6] that could consider this aspect. More specifically, we focused on adding new information, or new triples.

The update was constructed by removing graph fragments close to each other. First selecting random subjects, called seeds, and then performing a BFS with radius = 3. The resulting graph is composed not necessarily of connected components. The update's size depends on the starting graph's overall size because it is the result of a percentage factor, such as 10%, 20%, etc... At the end the starting dataset will be stripped of a part of it to be later added again. In this way we simulate a proper update.

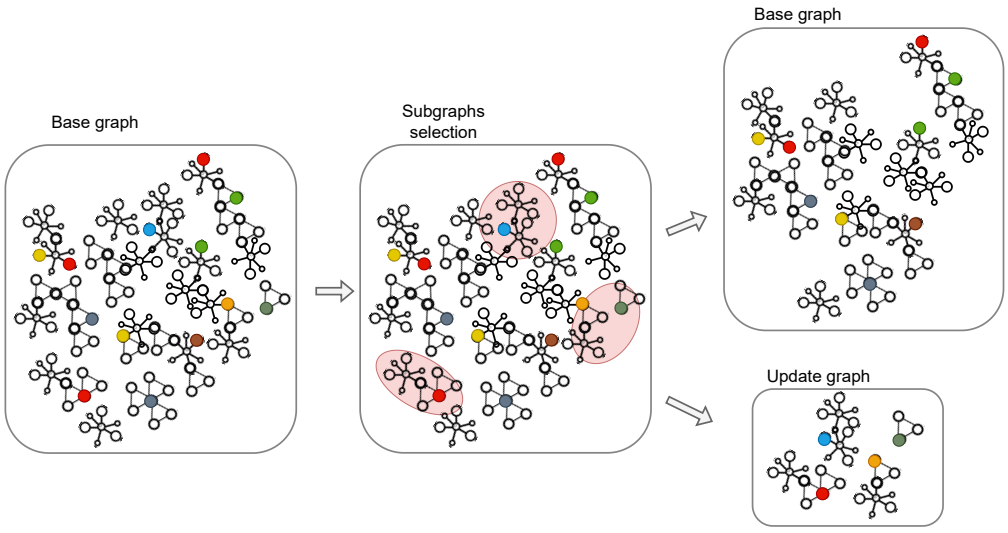

Figure 4.1: Visual representation of how the dataset to be considered as an update was created.

The selected triples are then marked and removed from the *base triple store* to be placed in another table called *updated triple store*. By *base triple store* we refer to the table representing the *base graph*, and by *updated triple store* the set of triples representing the new information, this table will act as the *update graph*. In the following section we will present the two approaches followed to reach the objective of this study. Section 4.1 will describe the first idea that came to mind, while Section 4.2 will present a different idea of handling updates based on the TSA algorithm.

4.1 Brute force approach

Before explaining Algorithm 2, some changes were applied to the original algorithm (Algorithm 1). More specifically lines of code have been added to create an auxiliary map to save the seed of a subgraph, that is the URI of the subject considered, together with the serial number of the cluster $(URI_i, cluster_i)$, i.e. $(< http://data.linkedmdb.org/resource/film/1374 >, 455)$ this means that specific film is the seed of the cluster nř 455. In this way, we can precisely pinpoint the subgraph to add new data.

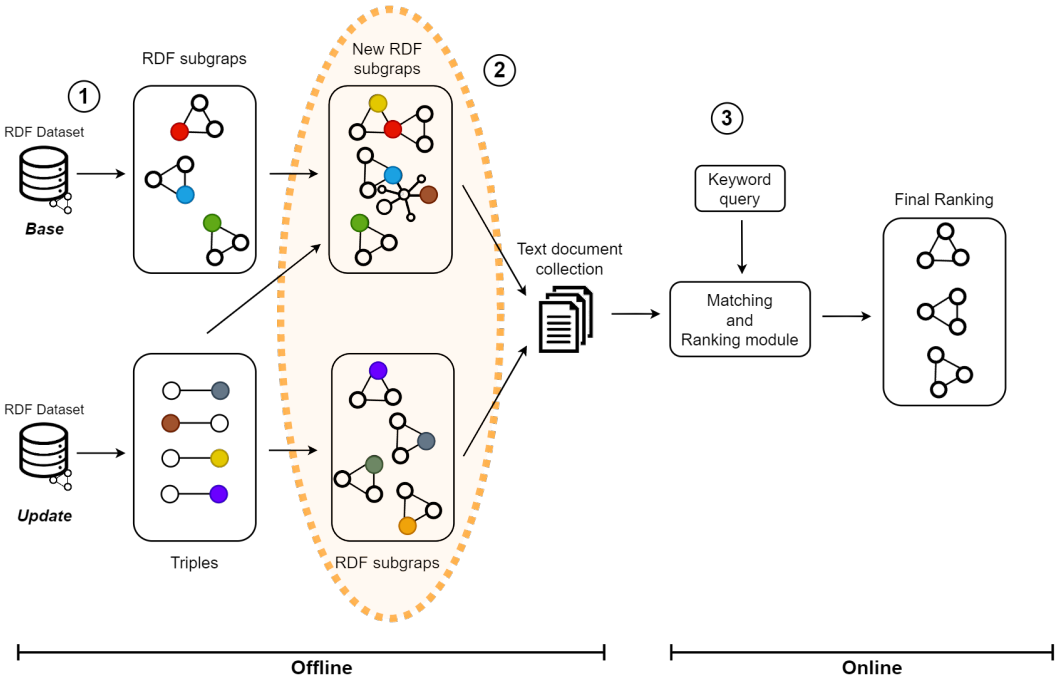

Figure 4.2: Overview of the brute force approach to handle the update

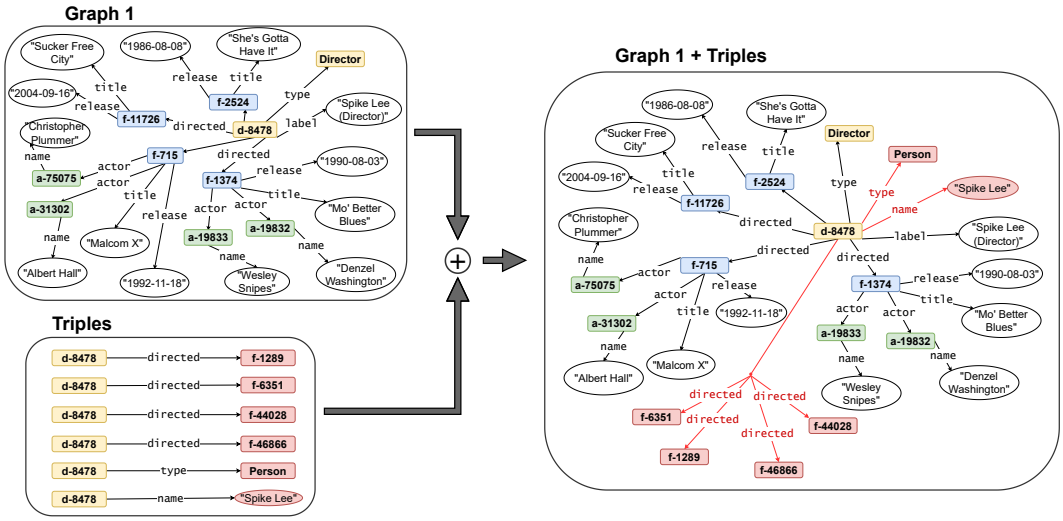

Figure 4.3: An example of how the BruteForce method works on an entity such as Spike Lee

Figure 4.2 represents the schema of the brute force method, the changes to the original system are highlighted: the first thing is to process the base dataset to generate the representative subgraphs, and after that the update dataset is considered. Alongside these new subgraphs are some essential metadata with which the subgraphs were produced, namely the couple $(cluster, numberofthecluster)$.

The second step involves considering the triples of the update. The triples inside the RDF graph are scanned one after the other and can follow two different paths. The first path involves a triple being added to a subgraph. This is because it shares the same subject as the seed node of that specific subgraph. The other path is the traditional one where a subgraph following the TSA algorithm is generated from that triple, more precisely from the subject of that triple. In this way, a syntactical cluster will be generated around the central seed, similar to what happened during the first graph base analysis.

In Figure 4.3 we can appreciate how the BruteForce algorithm works on a valid example. `Graph 1` is the subgraph of a director with the label Spike Lee exacted by the TSA algorithm from the base graph. Then we have a subset of the triples in which the subject is equal to the seed of the `Graph 1`, namely `d-8478`. When the BruteForce algorithm analyzes those triples it adds them to `Graph 1` liking the new properties to the central seed. In this example, the graph grows by 6 units: node `d-8478` gains four new *directed* statements, so 4 new films; then to the central node gains the property *type* and the property *name*.

The resulting graph has more information than the starting one, we now

know that director `d-8478` is a person and its name is Spike Lee, this fact could also have been inferred from the first graph but having a new property that directly specifies the name removes all doubt.

<DOCNO>383213</DOCNO> 8478 type director 8478 label Spike Lee (Director) 8478 made 11726 8478 made 1374 8478 made 2524 8478 made 715 11726 initial release date 2004-09-16 11726 title Sucker Free City 1374 initial release date 1990-08-03 1374 actor 19832 1374 actor 19833 1374 title Mo' Better Blues 2524 initial release date 1986-08-08 2524 title She's Gotta Have It 715 initial release date 1992-11-18 715 actor 31302 715 actor 75075 19832 name Denzel Washington 19833 name Wesley Snipes 31302 name Albert Hall 75075 name Christoper Plummer </DOC>

<DOCNO>383213</DOCNO> 8478 type director 8478 label Spike Lee (Director) 8478 made 11726 8478 made 1374 8478 made 2524 8478 made 715 11726 initial release date 2004-09-16 11726 title Sucker Free City 1374 initial release date 1990-08-03 1374 actor 19832 1374 actor 19833 1374 title Mo' Better Blues 2524 initial release date 1986-08-08 2524 title She's Gotta Have It 715 initial release date 1992-11-18 715 actor 31302 715 actor 75075 19832 name Denzel Washington 19833 name Wesley Snipes 31302 name Albert Hall 75075 name Christoper Plummer
8478 type Person 8478 director name Spike Lee 8478 made 6351 8478 made 1289 8478 made 44028 8478 made 46866 </DOC>

The system then proceeds to convert the subgraphs into text documents, and finally the keyword query is considered to retrieve the answers. The text on the left is what comes out after the graph-text conversion. The document lacks some relevant information, which is present in the right-hand document. Information such as the director's name, which is added later allows the right-hand document to gain a few positions in the final ranking since it is more relevant than the query.

The whole idea revolves around the concept in which a new triple t or a group of triples $(t_1, t_2, t_3, ..., t_n)$, from the *update* dataset, can bring new information to a subject s, which may already be present in the base graph B and thus be part of an already constructed subgraph $sbgr$. If the subgraph is not found this triple t becomes the starting point of a new subgraph, following the strategy of Alg. 1. The RDF subgraphs collection found in the *base* dataset can be enhanced by the new triples, but not all the subgraphs may be affected; at the same time new RDF graphs are produced from the triples left out. Brute-Force iterates over all triples t in the RDF graph U, that represent the update, then it takes the subject s and searches it in the map $\mathcal{L}$ where all the previous source nodes used by TSA (Alg. 1) are saved together with their serial cluster number, the cluster in which TSA placed them, for example if the node `" http://data.linkedmdb.org/resource/director/8478 "` that is Spike Lee is inside cluster 383213 than all the new triples regarding the director will be routed to that cluster. If the source node is found the cluster is modified adding the triple t. Otherwise, the source node becomes a new entry in the map $\mathcal{L}$ and a subgraph is created following the strategy of the TSA algorithm.

It is easy to spot the big drawback of this approach, the *for cycle* scans all the triple and in doing so there is the risk of spending more time in the update

processing than processing the base dataset. This suspicion is confirmed by the experimental data available in the next chapter.

The changes were applied only to the offline phase of the TSA algorithm, where the *virtual documents* are produced; the online phase, the part where the queries enter the process remained as the original one.

Algorithm 2 BruteForce

Input: RDF graph B, RDF graph U, map L, old $sbgr_{old}$,
Output: new representative subgraphs $sbgr_{new}$

```
1:  sbgr_new ← ∅
2:  for triple t in U do
3:      s ← t.getSubject()
4:      if s is in L then
5:          cluster R ← L.get(s) {R takes the value mapped to s}
6:          R.add(t)
7:      else
8:          c ← c + 1
9:          L.add(s,c)
10:         B' ← new Graph()
11:         while Q ≠ ∅ do
12:             v ← Q.dequeue()
13:             v.color ← black
14:             radius ← v.τ − 1
15:             for each u ∈ N⁻(s) do
16:                 if u ∉ S ∧ u ∉ T then
17:                     //Accessory node
18:                     G.addTriple((s,u))
19:                 end if
20:                 if u ∈ T then
21:                     // Terminal node
22:                     G.addTriple((s,u))
23:                     for each w ∈ N⁻(u) do
24:                         if w.isLiteral() ∨ (w ∉ S ∧ w ∉ T) then
25:                             G.addTriple((u,w))
26:                         end if
27:                     end for
28:                 end if
29:                 if (p(s,u) ∈ L) ∧ (radius > 0) ∧ (u.color ≠ black) then
30:                     if u ∈ S ∧ u ∉ T then
31:                         u.τ ← radius
32:                         Q.enqueue(u)
33:                         G.addTriple((v,u))
34:                     end if
35:                 end if
36:             end for
37:         end while
38:         sbgr_new.add(B')
39:         all the triples in B' are marked as visited and removed from U
40:     end if
41: end for
```

4.2 Semi TSA approach

This approach was designed as an alternative to BruteForce. The idea was to use what had already been developed for the original TSA system and apply it directly to the dataset used as an update. Algorithm 3 shows how the system detects subgraphs. It starts by tanking in input the RDF graph U, the graph that acts as the update. The statistics about the source and terminal set calculated in the base graph B processing are reused here. This is done to ensure that a node is classified in one of the two sets, source or terminal, with a higher probability and that as a result the node is used for the creation of the virtual documents. On the other hand, that precious information might have been lost.

Another critical detail is defining the neighbors of a node extracted from the queue Q. The search for the neighbors is also extended to the base graph and is not confined to just the update graph. Any duplication generated, which is almost certain since the neighbor search is extended beyond the boundaries of the update graph, will be eliminated in the online phase of BM25 and VDP where only useful triples remain and duplicates are suppressed.

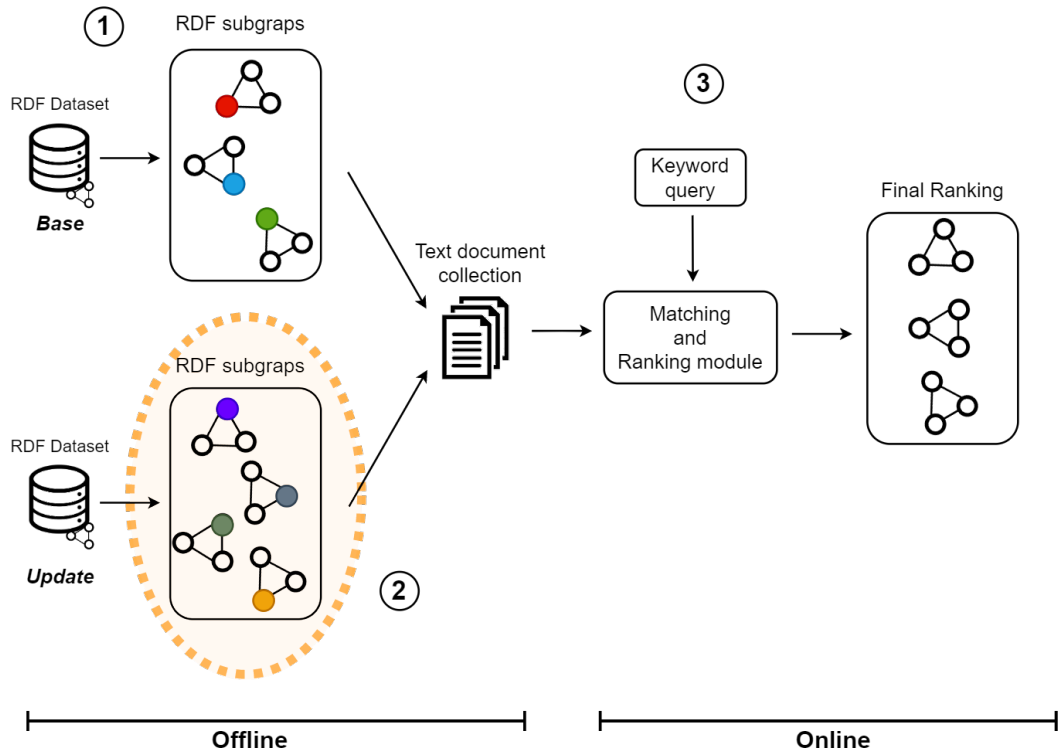

Figure 4.4: Overview of the Semi TSA approach to process the update

Figure 4.4 shows how this system works: the computation done for the base graph B is replicated here. This also includes the source and terminal node computation part, because in this way we are able to capture the new data present in the update, like new actors, film or additional data to enrich the base graph. The subgraphs produced by the update dataset are converted and added to the text document collection. The next steps are the same as those followed in the original paper [6]. In the online part, the search will be performed on all documents produced, either in those built from the base graph and those new ones introduced by the update analysis.

The main idea is to create new virtual documents by considering the update as an addition to the original system, which is effectively the nature of the update itself. The update is a starting point for creating new documents that are richer in information than those created using only the base graph. This is because the neighboring nodes, and any literals, that will make up the cluster come from both the base graph and the update graph. Here the statistical computation is confined only to the update, this enables a quicker processing time and reduced complexity. The total complexity, however, remains very similar to the original complexity obtained by running the TSA+BM25 or TSA+VDP algorithm on the complete graph.

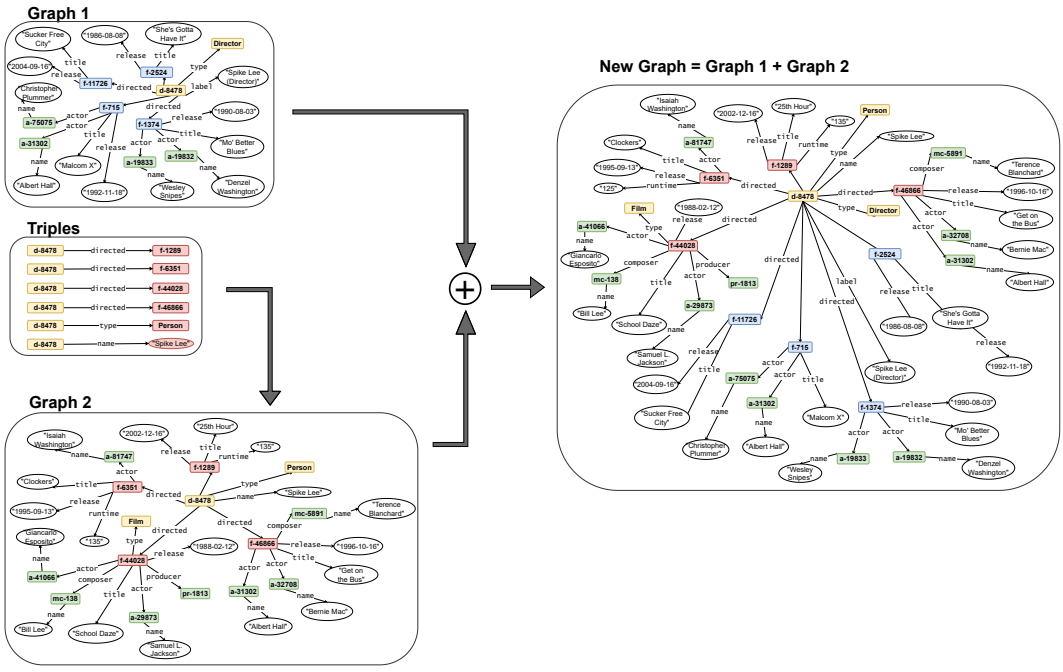

Figure 4.5: An example of how the semiTSA method works on an entity such as Spike Lee

In Figure 4.5 there is a concrete example of how the semiTSA system works. The situation is the same as in the example considered earlier to illustrate the operation of the BruteForce system. Consider the case in which the triples of the update to be analyzed include the same 6 seen before. This system, as can be appreciated from the figure, produces a second graph, `Graph 2`, independent of `Graph 1`. Observing the algorithm 3 it seems that this fact is missing but Graph2 is the graph that would be created if we restricted the TSA algorithm only to the update database U. Instead, by allowing freedom to explore the base graph B again the results is the union of the two because the triples of graph B are again explored in the same way as the original algorithm.

This new graph looks very similar to `Graph 1`. Looking at this graph we can immediately conclude that the seed is node d-8478, it is a person type entity and it is named Spike Lee. Around him other nodes develop, the same ones we had in the BruteForce case. Compared with BruteForce, the red nodes are explored again with a technique similar to BFS. There is much more information here: the title of each thing Spike Lee directed, when the product or movie was released, and which actors, with their names, participated in the work. There are also some durations of the movies. All this new information is then transformed into a text document. Then `Graph 1` and `Graph 2` are merged together and the resulting graph becomes part of the set of representative subgraphs, to later be converted into a text document.

<DOCNO>383213</DOCNO> 8478 type director 8478 label Spike Lee (Director) 8478 made 11726 8478 made 1374 8478 made 2524 8478 made 715 11726 initial release date 2004-09-16 11726 title Sucker Free City 1374 initial release date 1990-08-03 1374 actor 19832 1374 actor 19833 1374 title Mo' Better Blues 2524 initial release date 1986-08-08 2524 title She's Gotta Have It 715 initial release date 1992-11-18 715 actor 31302 715 actor 75075 19832 name Denzel Washington 19833 name Wesley Snipes 31302 name Albert Hall 75075 name Christoper Plummer </DOC>

<DOCNO>383213</DOCNO> 8478 type director 8478 label Spike Lee (Director) 8478 made 11726 8478 made 1374 8478 made 2524 8478 made 715 11726 initial release date 2004-09-16 11726 title Sucker Free City 1374 initial release date 1990-08-03 1374 actor 19832 1374 actor 19833 1374 title Mo' Better Blues 2524 initial release date 1986-08-08 2524 title She's Gotta Have It 715 initial release date 1992-11-18 715 actor 31302 715 actor 75075 19832 name Denzel Washington 19833 name Wesley Snipes 31302 name Albert Hall 75075 name Christoper Plummer
8478 type Person 8478 director name Spike Lee 8478 made 6351 6351 title Clockers 6351 initial release date 1995-09-13 6351 runtime 125 6351 actor 81747 81747 name Isaiah Washington 8478 made 1289 1289 initial release date 2002-12-16 1289 title 25th Hour 1289 runtime 135 8478 made 44028 44028 initial release date 1988-02-12 44028 type Film 44028 actor 41066 44028 composer 138 44028 title School Daze 44028 actor 29873 44028 producer 29873 name Samuel L.Jackson 138 name Bill Lee 41066 Giancarlo Esposito 8478 made 46866 46866 composer 5891 5891 name Terence Blanchard 46866 initial release date 1996-10-16 46866 title Get on the Bus 46866 actor 32708 32708 name Bernie Mac 46866 actor 31302 31302 name Albert Hall </DOC>

The information that the new graph brings to the system is superior to that generated by the BruteForce algorithm. Here are the titles of the films and the names of the actors who worked on them. To a possible question "What movie has Spike Lee directed?" The documents produced by the semi-TSA algorithm

are more accurate than those of BruteForce because they contain all the keywords required to answer the question. As a result, they are ranked higher in the final ranking.

Algorithm 3 Semi TSA

Input: RDF graph B, RDF graph U, map L_{new}, source set S_{new}, terminal set T_{new}, source set S_{old}, terminal set T_{old}
Output: new representative subgraphs $sbgr_{new}$

```
1: sbgr_new ← ∅
2: Q ← ∅
3: for each s ∈ S do
4:    if s.color = white then
5:       Q.enqueue(s)
6:       s.τ ← τ
7:       G ← new Graph()
8:       while Q ≠ ∅ do
9:          v ← Q.dequeue()
10:         v.color ← black
11:         radius ← v.τ − 1
12:         //search in both graph B and graph U
13:         for each u ∈ N⁻(s) do
14:            if u ∉ S ∧ u ∉ T then
15:               //Accessory node
16:               G.addTriple((s,u))
17:            end if
18:            if u ∈ T then
19:               // Terminal node
20:               G.addTriple((s,u))
21:               //search in both graph B and graph U
22:               for each w ∈ N⁻(u) do
23:                  if w.isLiteral() ∨ (w ∉ S ∧ w ∉ T) then
24:                     G.addTriple((u,w))
25:                  end if
26:               end for
27:            end if
28:            if (p(s,u) ∈ L) ∧ (radius > 0) ∧ (u.color ≠ black) then
29:               if u ∈ S ∧ u ∉ T then
30:                  u.τ ← radius
31:                  Q.enqueue(u)
32:                  G.addTriple((v,u))
33:               end if
34:            end if
35:         end for
36:      end while
37:      sbgr.add(G)
38:   end if
39: end for
40: return sbgr
```

5

Analysis

In this chapter we report and reason about the experimental results of the modified TSA+BM25 and TSA+VDP to handle updates, described in chapter 4, comparing them with the ones from Dosso e Silvello[6].
The experiments were performed on two databases, LinkedMDB and a subset of IMDB. Only LinkedMDB(or LMDB) is a native RDF dataset, IMDB is shared as a relational database, so it was converted into a RDF dataset. Both are datasets about movies or shows and everything that revolves around them. LinkedMDB is composed by more then 6 million triples, whereas the size of the IMDB subset is 30 million. To ensure that the answers to our queries were there, ground truth triples were added to the subset. All tests were performed on a laptop equipped with Windows 11, i7-10850H @ 4.00 GHz, 32GB of RAM. The datasets were imported in PostgreSQL, creating a table with three columns, "subject", "predicate" and "object" and auxiliary indexes. Blazegraph, a native application to manage RDF graphs, was used to generate only the ground truths. Thanks to the available computing power and system efficiency a few parameters have been changed over the original system. In particular the value l of the first ranked graphs that are passed to the merging function, and the value n of the VDP pipeline to build the big query graph to retrieve the answers.

The following performance results, referring to the four different update percentages, are the average of three distinct runs, trying to create each time a unique update set for the experiment. The term baseline in the following chapter will represent the run and numerical results obtained using the original TSA+BM25 and TSA+VDP systems. Section 5.2 will present results obtained with a brute force technique, while section 5.3 has results obtained considering the update as a separate thing using a more quick virtual document mechanism.

5.1 Update construction

The update was constructed by relying entirely on PostgreSQL, specifically the BFS exploration was performed on subjects selected at regular intervals in the triple_store table. Below we have reported the code. The first question mark allows you to choose the sampling interval; the second one limits the results. With `DISTINCT` we are sure to generate a set. This could be seen as a suboptimal method of selecting nodes, but after several attempts, this strategy was adopted to reach the necessary number of triples to be excluded. This does not preclude alternative mechanisms from being used in the future.

```
SELECT DISTINCT t.subject_, t.id_  FROM
(
    SELECT subject_, id_, ROW_NUMBER() OVER (ORDER BY id_) AS rownum
    FROM this.schema.triple_store_base
) AS t
WHERE t.rownum % ? = 0
ORDER BY t.id_ limit ?;
```

Code 5.1: SQL command to select the subject seeds

Database name	Dimension	Num of queries
LinkedMDB	6.1 millions	50
IMDB	30 millions	50

Table 5.1: Name of the dataset used and their dimension

5.2 Brute force strategy

Table 5.2 shows my first attempt at handling the update the basic and brute force mechanism on both datasets. The poor results of tb-DCG highlight how the technique adopted is not very effective. In LinkedMDB, at 10% the loss compared to the baseline is 36% and gets worse as the overall update percentage is increased on both BM25 and VDP systems. Such results could be explained by the fact that the update triples were not inserted as well as they could have been by processing the whole graph at once, this led to a decrease in the relevance of the graph and a consequent reduction in the overall tb-DCG. Since tb-DCG is affected by the size of the ground truth, we can infer that the resulting graph on average is

Dataset	percentage of update	Systems	tb-DCG	recall	prec@1	prec@5	num of clusters
LinkedMDB	0 %	BM25	0.183	0.855	0.000	0.006	471.0 k
		VDP	0.537	0.861	0.034	0.032	
	10 %	BM25	0.117	0.591	0.000	0.003	518.1 k
		VDP	0.343	0.394	0.025	0.022	
	20 %	BM25	0.113	0.569	0.000	0.002	536.6 k
		VDP	0.315	0.345	0.038	0.023	
	30 %	BM25	0.107	0.545	0.002	0.000	544.8 k
		VDP	0.388	0.424	0.063	0.046	
	40 %	BM25	0.118	0.588	0.000	0.000	538.6 k
		VDP	0.439	0.487	0.053	0.048	
IMDB	0 %	BM25	0.152	0.317	0.121	0.061	933.9 k
		VDP	0.255	0.281	0.031	0.016	
	10 %	BM25	0.057	0.126	0.066	0.013	917.5 k
		VDP	0.224	0.234	0.024	0.013	
	20 %	BM25	0.049	0.095	0.026	0.005	897.0 k
		VDP	0.251	0.258	0.034	0.022	
	30 %	BM25	0.037	0.072	0.040	0.006	878.5 k
		VDP	0.196	0.205	0.026	0.016	
	40 %	BM25	0.034	0.067	0.023	0.006	866.3 k
		VDP	0.161	0.177	0.022	0.018	

Table 5.2: Results obtained with the brute force approach

smaller than it should be. This may be combined with a consequential difficulty for BM25 to sort the graphs by relevance. Here the precision performance is very similar to the baseline in both BM25 and VDP, and a little better in prec@5 with VDP in the case of 30% and 40%. Maybe those numbers may be affected by the number of clusters which is steadily increasing as the update rate increases.

With VDP at 30% and 40% tb-DCG values are a little higher but the main reason could be the dimension of the update: the new clusters are more precise, with fewer useless triples, and are ranked better. The recall cannot come close to the excellent result obtained by the baseline, but it did not change, increasing the update percentage, meaning that at least half of the triples are retrieved with the BM25 pipeline. VDP because it only considers the first 120 results from BM25 gets better tb-DCG, thanks to the embedded pruning and re-ranking process but lower recall, in line with baseline outcome.

Shifting to IMDB, we notice a difference in the results with this dataset; the results on the LinkedMDB are relatively close to the baseline and so a little unexpected, for a such simple strategy, the ones obtained on IMDB shows that this method is not suited for handling updates on a RDF keyword search system on that particular dataset structure. Looking at tb-DCG the loss with respect to the BM25 baseline, at 10%, is 62.5%, at other percentage levels the value

gradually decreases; the difference between 30% and 40% is marginal. VDP demonstrates how the ranking and results obtained at BM25 stage are crucial for tb-DCG: from 20% onward we can see how progressively more than 30 points are lost at each step. This means that the ranking of BM25 is sub-optimal and gets worse as the update rate increases, plus the generated graphs carry many irrelevant triples. One thing to notice is how the recall in the VDP pipeline continues to have good results until 30%, then it gets worse.

Precision values are low but constant throughout the update percentages. With some surprise, the recall values of VDP are higher than BM25, this can be explained by the approach of VDP and the nature of the database. Here the pruning increase the SNR and the retrieved graphs are bigger and contain more relevant triples, but the lower performance of recall compared to the baseline may be due to the high connectivity of the dataset and also one of the key factor that differentiate the progression in the number of clusters between the two datasets: while in LinkedMDB there is an increase of the number of the cluster as the percentage grows, in IMDB the number decreases significantly, many triples are inserted in already created cluster penalizing the creation of new and more accurate subgraphs, as previous results show. The rate at which we find a relevant, or partially relevant, result in the first position is consistent with the baseline and in all update cases.

In IMDB, the loss in performance moving from 10% update to 40% update is greater than 34%, which also happens in LMDB where the difference is 35%. Recall sinks heavily relative to the baseline, in particular in IMDB.

5.3 Semi TSA strategy

5.3.1 Single update

Dataset	percentage of update	Systems	tb-DCG	recall	prec@1	prec@5	num of clusters
LinkedMDB	0 %	BM25	0.183	0.855	0.000	0.006	471.0 k
		VDP	0.537	0.861	0.034	0.032	
	10 %	BM25	0.128	0.642	0.002	0.003	495.6 k
		VDP	0.485	0.517	0.021	0.021	
	20 %	BM25	0.135	0.703	0.000	0.002	499.7 k
		VDP	0.499	0.538	0.022	0.020	
	30 %	BM25	0.141	0.721	0.000	0.004	505.8 k
		VDP	0.553	0.581	0.025	0.022	
	40 %	BM25	0.160	0.733	0.000	0.005	499.7 k
		VDP	0.520	0.572	0.021	0.020	
IMDB	0 %	BM25	0.152	0.317	0.121	0.061	933.9 k
		VDP	0.255	0.281	0.031	0.016	
	10 %	BM25	0.118	0.294	0.083	0.014	929.8 k
		VDP	0.228	0.247	0.041	0.025	
	20 %	BM25	0.095	0.287	0.041	0.011	923.6 k
		VDP	0.228	0.262	0.039	0.025	
	30 %	BM25	0.109	0.307	0.045	0.012	919.5 k
		VDP	0.198	0.235	0.037	0.032	
	40 %	BM25	0.116	0.316	0.045	0.017	915.4 k
		VDP	0.188	0.231	0.036	0.023	

Table 5.3: Results obtained with TSA pipelines and a variable update percentage

Table 5.3 shows the results obtained by the system under various conditions by the two modified pipelines. Starting with LMDB, it is clear how the two systems behave in different ways, the drop in performance with tb-DCG is less with VDP than with BM25. With an update of 10% VDP loses only 10% compared to the 30% of BM25 relative to the baseline. This could be explained by the increase in the number of clusters; more clusters mean that the information is more scattered, so the BM25 pipeline has a hard time merging and ranking the solution graphs.
We can observe a strange behavior, tb-DCG value increases as the percentage of the update increase; at 40% the value is only 12% less than the baseline. The same thing is replicated by VDP, at 40% surprisingly the value is even higher than the baseline. The same effect appears considering recall, but here the results tend to stay on the same level or rise a little on every update percentage level.

The improvement could be due to the effect of "new" information brought by the update, with the latter being big enough to create "virtual documents" similar to the baseline. Considering an update as a single entity, virtually separated from the base dataset, it produces more virtual documents and thus the number of clusters. Still, it remains more or less constant in every situation, despite increasing the triples being considered an update. The number of clusters directly influences the online phase, we noticed that some queries, during the VDP pipeline, had a bigger *query graph*, the graph from which the system starts a BFS and retrieves the best candidates to be passed to the following method. For instance, query 1 *"Francis Ford Coppola director title"* of LinkedMDB had a query graph size(number of triples), which is generated in VDP before pruning, of 5738 with an update of 10%, obtaining 0.63 as tb-DCG; the query graph size grew at 6646, 7378 and 7626 for the following percentages and in all those cases tb-DCG was equal to 1, the identical metric value obtained from the baseline. In this case, we can assume that in the 20% update and above, the graphs are closer to their baseline counterparts. In all cases, all the relevant triples are retrieved but in the first update case those are not ranked in the first two positions (then it starts the discounted mechanism) and so the value is lowered a bit.

The variability of the dimension of the graph size is confirmed by query 23 *"actor director title Dennis Hopper"* where the values ranges from 1295 at 10% to 2560 at 40%. Still, at the end, the tb-DCG value is maximum while the precision is only 0.03 similar to the baseline.

Looking at recall, we observe a decrease we expected. Here BM25 is the top-performing system, at 10% the recall loss compared to the baseline is over 23% and regain some points as the update percentage increase. We may think that this is due to more precise and less noisy clusters that TSA can build, with higher percentages of updates, we get closer and closer to the initial condition in fact, at 40% BM25 obtains 20% more points compared to the 10% update. A similar thing happens with VDP after the difference between 10% and 40% settles to 10% with the advantage of the latter.

As we can see from the table, precision values are closer to the baseline on all occasions, this suggests that the new subgraph may not be optimal to have a good performance in the other metrics but here they maintain useful triples and are placed in the top positions.

Moving to IMDB the conclusions are similar, VDP is the system that retains

more performance over all updates scenarios. The pruning and re-ranking techniques seem to be effective, but as already observed by Dosso and Silvello [6], the structure and nature of the database have an impact on the results. Here tb-DCG decreases moving from 10% to 40%, compared to LinkedMDB where there was an opposite trend. The loss of BM25 compared to the baseline is around 27% at 10% and does not improve as the update percentage increases. VDP has the same behavior, until 20% tb-DCG the loss is around 10% but then it increases till 26%. The decreasing number of clusters evidences the features of IMDB as the update rate increases even with respect to the baseline, contrary to what we would expect. Presumably the update separated the database in a way that favored higher connectivity so the subjects cluster more easily.

Recall instead have comparable outcomes during all tests and with some surprise the results are near the baseline; a complete different results compared to LinkedMDB. Closer to the baseline are the results obtained in the precision area especially with VDP. BM25 instead loses only 7.2% from a 10% update but it increases a bit when the update size increases. Here VDP has lower results compared to BM25.

Looking at the precision@1 values BM25 loses up to 66% at 20%; VDP, thanks to the re-ranking approach and intermediate steps with the query graphs, manages to have higher and stable results in all the occasions, superior to the baseline. Prec@5 proposes the same trend observed above, the VDP is the system that gets the best results.

The same thing happens in IMDB where despite queries have a query graph each time of a different dimension, the results are constant, like in query 12 *"name Orson Welles profession director known for titles cast actor"*. At 10% the query graph size is 1116, then in the other cases 1426, 1519 and finally 1267. The difference of the dimensions of the query graphs is a signal of how the virtual document selected by the BM25 part are bigger and probably how they retain needless triples that are removed in the next step.

5.4 Comparison between methods

Here we compare the two different approaches, BruteForce and TSA. As seen in the previous sections, IMDB enhances the differences between methods, mostly caused by its dimension and structure. Considering *tb-DCG* in Figure 5.1 it is visible how semi-TSA, in green, with the BM25 pipeline, with a 10% update,

manages to find an answer that in most of the cases is higher or even the only one available compared to the first system. Not in all queries the results are great, the queries in the second half get good values, and you see the advantage of the second approach. From Figure 5.2, which represents the difference in performance in the two systems for each query, we can appreciate that only four times BruteForce is capable of finding a better solution with an advantage of less than 0.2 points. In some query reach 0.4 points. When there are no bars it means that the two systems are equivalent. On the other hand, this image highlights the excellent results obtained by semi-TSA. In general we can say that semi-TSA it is best suited to handle this case.

The differences are more clear looking at higher update percentages of 20% or 40%. Figures 5.3 and 5.4 highlight the deteriorating performance of the first strategy. Values from tables 5.2 and 5.3 confirm this aspect. For instance, at 40% queries like 28, 43 or 46 flipped in favor of semiTSA. The semi-TSA gains more than three times in tb-DCG with respect to the BruteForce approach looking at the 40% update values in BM25, but also at other levels the advantage of semi-TSA is clear.

VDP instead shows a slightly different situation. Here the differences between updates are already evident starting between 10% and 20% updates as depicted in Figures 5.7 and 5.8. If at one level of the 10% BruteForce and semi-TSA seem to perform in the same way, in fact the numerical difference is minimal, because the advantages on some queries are eliminated on others. In the case of 20% BruteForce surprises and gets a better result than semi-TSA. From the query-to-query graph in Figure 5.8 the advantage is in queries 15, 16, 17, 41, 45, 46. The particular advantage on these queries is maintained also in the following percentages, but the values in the other queries drop, so semi-TSA regains first place. Differences in VDP pipeline do not stand out, tb-DCG values are more constant and higher in the semi-TSA approach, at 40% update semi-TSA gains roughly 15%; the result of BruteForce at 20% is impressive.

Recall is the second metric that we want to compare, both Figures 5.9 and 5.10 show the values under BM25, there it is highlighted how BruteForce method practically always loses. We can see from Figure 5.10 that with 20% semi-TSA method is the top one in all queries in particular in the last twenty and differences from 10% are substantial. The brute force strategy reveals its limit and it is not well suited to process and scale updates in an RDF environment, at least with this implementation. The brute-force method loses up to 67% compared to

semi-TSA in the same 20% update situation and in the other cases it does not have great results. Recall in BruteForce VDP degrades as the update increases, semi-TSA on the other hand proves much more flexible and even improves by a few points. The performance delta is not as sharp as with BM25. VDP has the characteristic of decreasing the differences between the two systems noticeably, and this is inferred by the previous graphs about VDP. One example is given in Figure 5.11, the advantage of VDP is only 8 queries.

In both approaches the precision is a metric greatly affected by the update size. In both cases above an update size of 10% the precision@1 is halved. Semi-TSA, in the VDP pipeline, performs better than the baseline, whereas BruteForce stands at lower values.

BruteForce produces lower number of cluster compared to TSA, while semi-TSA has a little variability, BruteForce instead steadily reduces the value.

Proceeding with an analysis similar to that done for IMDB we examine LinkedMDB. In LMDB, the brute-force strategy is not effective, the tb-DCG loss compared to the semi-TSA is more than 24%, at 30% with BM25 and in the Figure 5.13 this difference is clear with minor exceptions. Strangely, while in BruteForce there is a decrease in values as the percentage increases, in semi-TSA the opposite happens. VDP pipeline even shows how the differences between the two approaches, in this case with a 30% approach, there are no intermediate measures: either the system perfectly detects the answer, so with a value of 1, it fails, or as happens here specifically the two systems are equivalent like we see in Figure 5.14 and 5.15. VDP with the Semi-TSA approach performs more than 43% better than BruteForce in these conditions. With an update size of 10% the advantage is similar, 41%. With this database VDP pipeline outperforms BM25.

Looking at the recall semi-TSA maintains an excellent level of performance. BM25 proves to be the better system of the two under update conditions if we aim to optimize this parameter. In fact, the scores never fall below 60%. The final values improve, as the size of the update increases, up to 14%. In contrast, BruteForce does not scale correctly and gets worse as the size of the update increases. The loss transitioning from BM25 to VDP is similar for the two methods but in the end it is still semi-TSA to maintain first place. Here we find the characteristics of the query-by-query analysis seen for IMDB; values of VDP like those seen before in IMDB confirm its trend. We will not report other update size analysis because all the graphs will be similar to each other and they

would not bring any further insight.

The precision in the LMDB is extremely similar in both approaches. Perhaps the update handling is not optimal and should be revised to improve this aspect, BM25 in nearly all cases do not have any relevant document in the first position. Some things change for VDP but this it is primarily thanks to the second ranking done in the pipeline.

In VDP the advantage ranges from 12 up to 42%(when considering an update of 30%) The recall performance of semi-TSA is significantly higher than brute-force in every scenario.

5.4.1 Execution times

Looking at the execution times reported in table 5.4 we notice how the semiTSA approach is quicker in every situation, and constant with respect to the BruteForce algorithm and the advantage grows as the size of the update becomes larger. The BruteForce approach is greatly affected by the increase in triples as the update ratio grows, because it does not create a new virtual document immediately but instead, it tries to look for a premade document in which to insert the additional information, this observation is valid in both datasets. In fact we can see that even at 10% the update processing time is higher than the base; the fact remains valid for all the conditions, giving us a hint that this idea not only performs worse than its counterpart. It is also worth mentioning that most of the offline time is spent on the updating time as the percentage of the update increases, the other parts like document creation and indexing tend to be constant because they use state of art techniques. Focusing to semiTSA we can see that in LMDB until 30% the time to handle the update is lower than the base, while it is superior at 40%. In IMDB the moment in which the update processing time is greater than the base processing, happens with 30%. The difference might be due to the size but also because IMDB has a more connected structure in which semiTSA, inheriting everything from the original system, has to dive. Looking at Figure 5.18 one thing to note in BruteForce is the distinct increase in running time in the transition from 20% to 30% in both datasets; again in either dataset we can point that the execution time for semi-TSA at 40% is even lower than BruteForce at 10% . semiTSA instead maintains a stable and linear execution time in every situation doubling the size of the dataset also duplicates the execution time. The table shows the two trends: the semi-TSA

maintains a reasonable processing time for the update set in both datasets, while BruteForce takes 4 to 5 times as long. From those data semiTSA seems to be a scalable strategy and it is the right one to process big data. The fact should not be surprising and confirms how this innovative approach holds great potential in the field of keyword search by fitting into the dense picture presented.

Dataset	percentage of update	Systems	Total time (min)	Base (min)	Update (min)
LinkedMDB	0 %	Baseline	23	10	/
	10 %	Brute force	32.71	10.47	10.76
		semiTSA	25.11		2.39
	20 %	Brute force	36.20	9.68	16.68
		semiTSA	26.12		4.28
	30 %	Brute force	45.04	8.80	25.55
		semiTSA	26.32		7.00
	40 %	Brute force	48.47	6.32	33.04
		semiTSA	23.25		8.25
IMDB	0 %	Baseline	102	50	/
	10 %	Brute force	144	39.49	55.30
		semiTSA	96		10.12
	20 %	Brute force	168	34.10	93.14
		semiTSA	96		20.78
	30 %	Brute force	204	27.25	140.75
		semiTSA	90		30.21
	40 %	Brute force	240	22.82	184.78
		semiTSA	96		42.52

Table 5.4: Offline time execution of the different systems

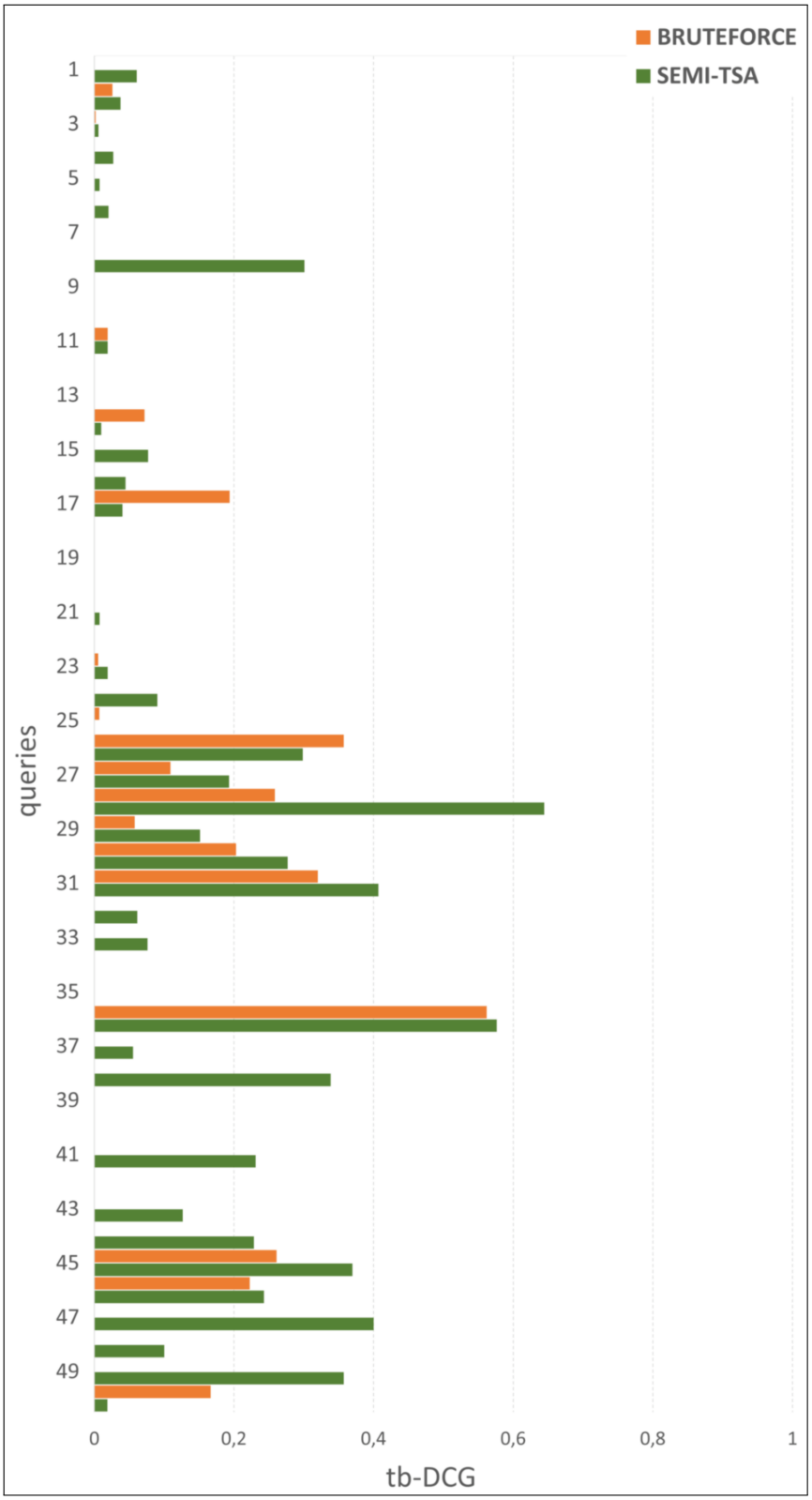

Figure 5.1: Comparison between methods using BM25 in IMDB tb-DCG at 10%

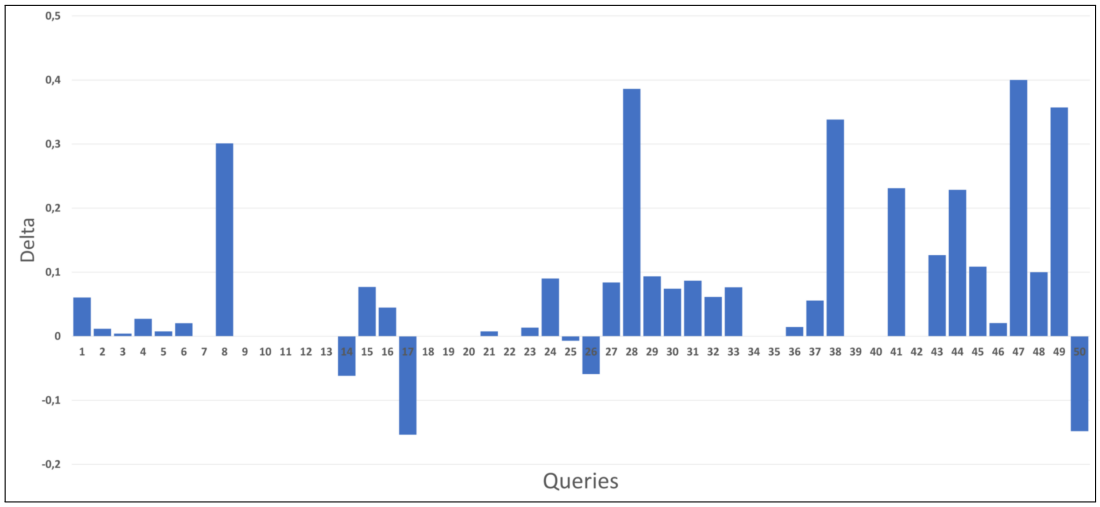

Figure 5.2: Differences in tb-DCG between BruteForce and semi-TSA in IMDB 10% update using BM25 pipeline

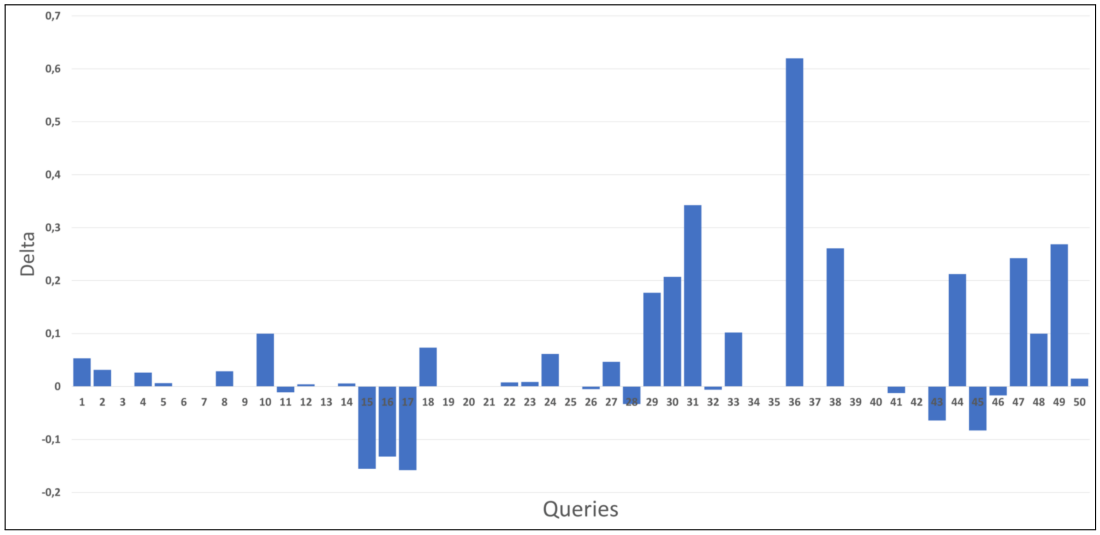

Figure 5.3: Differences in tb-DCG between BruteForce and semi-TSA in IMDB 20% update using BM25 pipeline

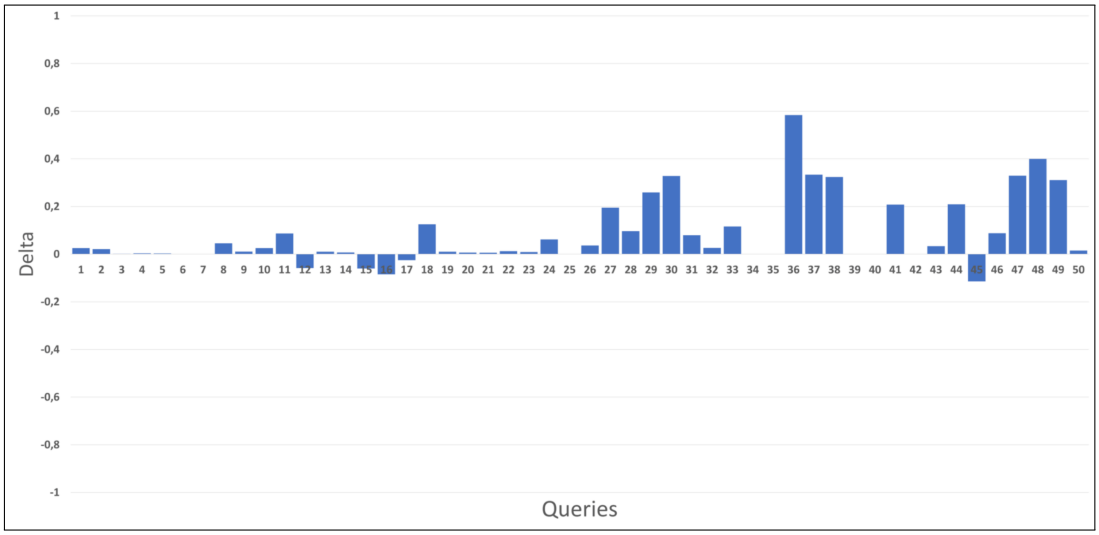

Figure 5.4: Differences in tb-DCG between BruteForce and semi-TSA in IMDB 40% update using BM25 pipeline

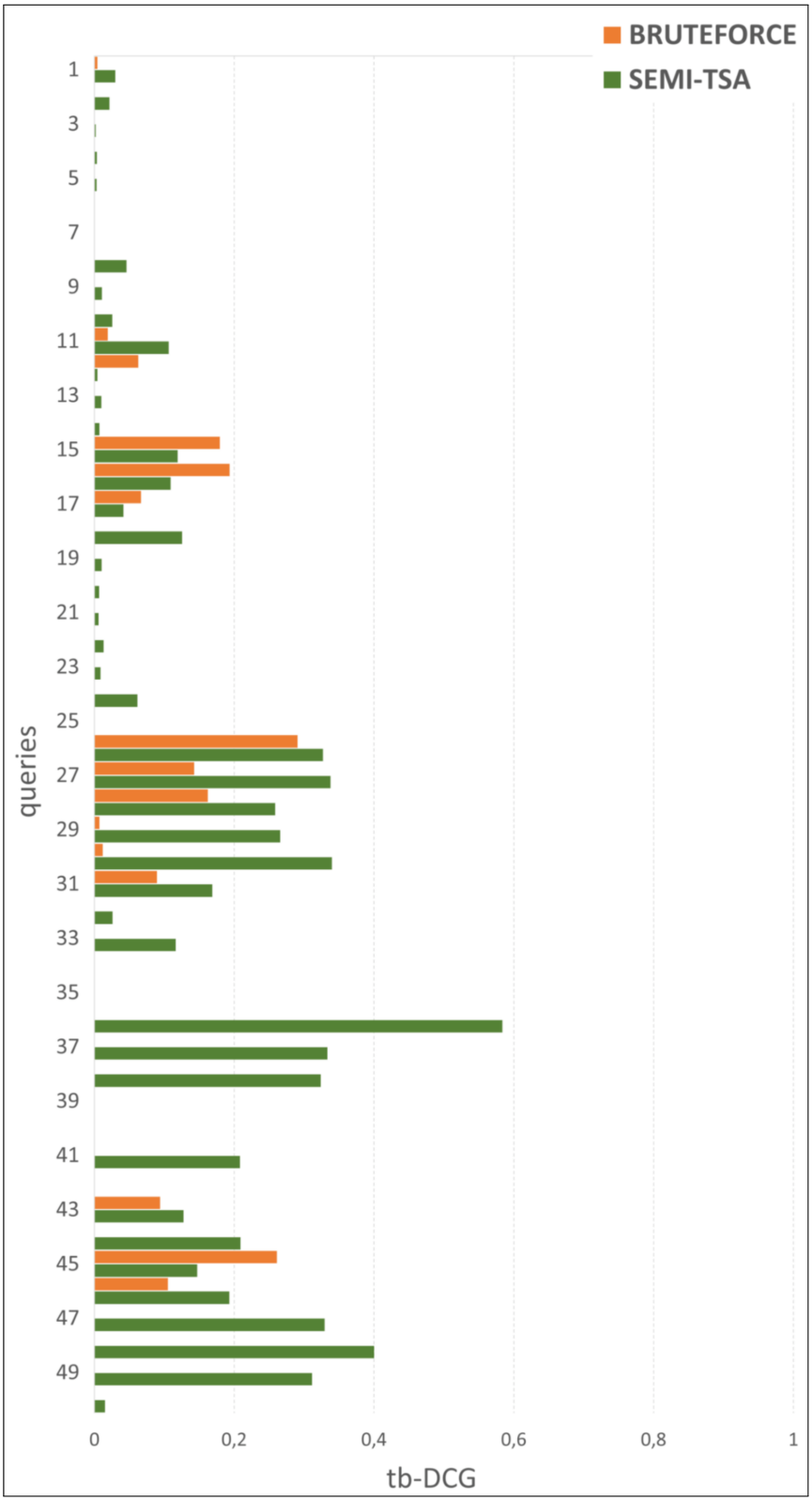

Figure 5.5: Comparison between methods using BM25 in IMDB tb-DCG at 40%

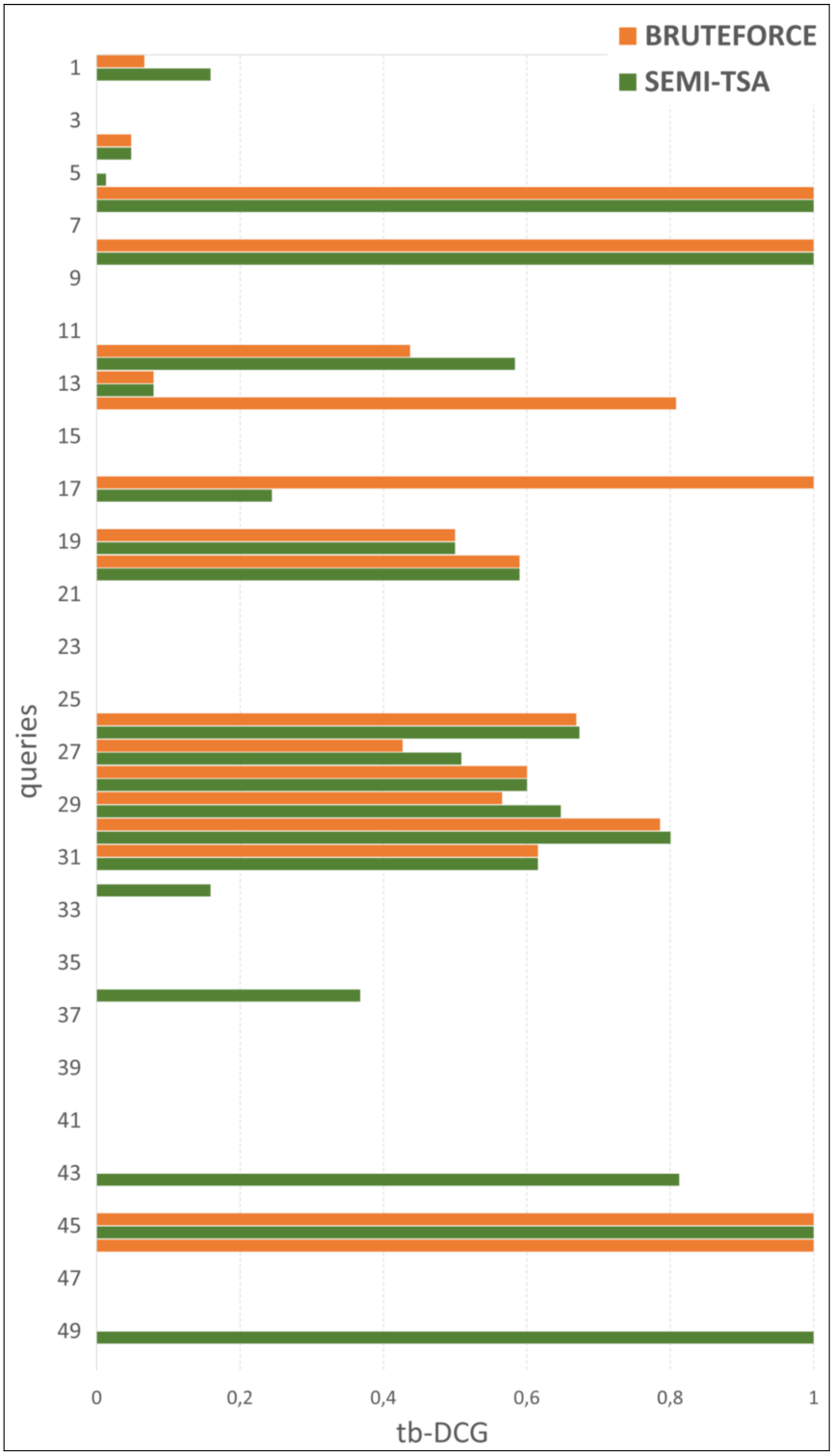

Figure 5.6: Comparison between methods using VDP in IMDB tb-DCG at 10%

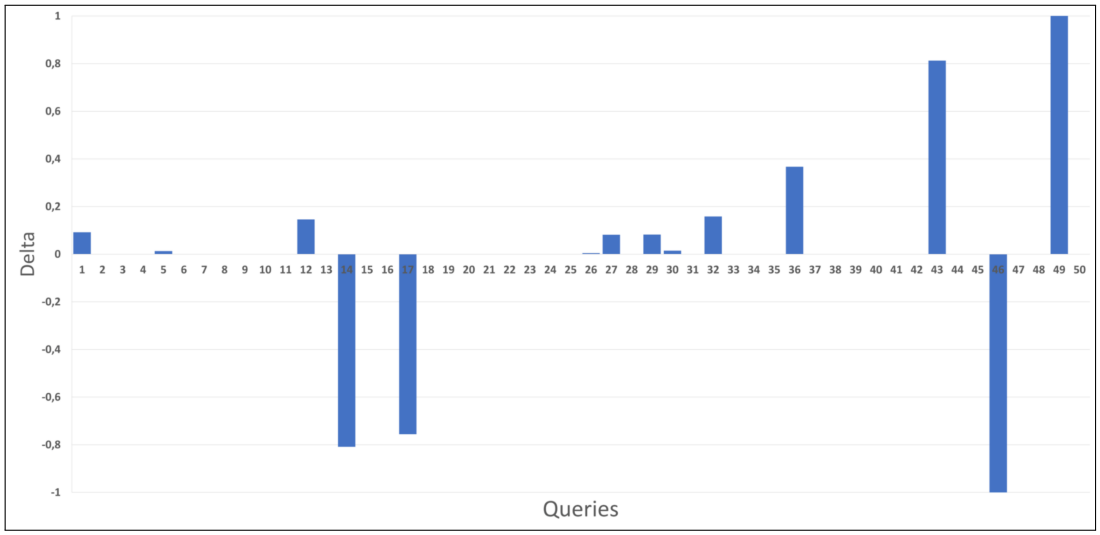

Figure 5.7: Differences in tb-DCG between BruteForce and semi-TSA in IMDB 10% update using VDP pipeline

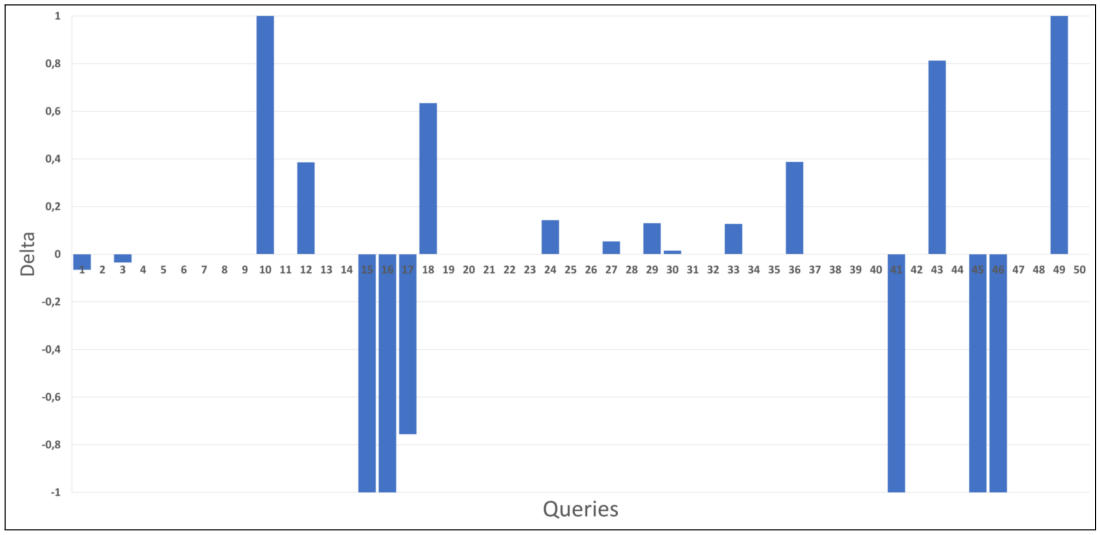

Figure 5.8: Differences in tb-DCG between BruteForce and semi-TSA in IMDB 20% update using VDP pipeline

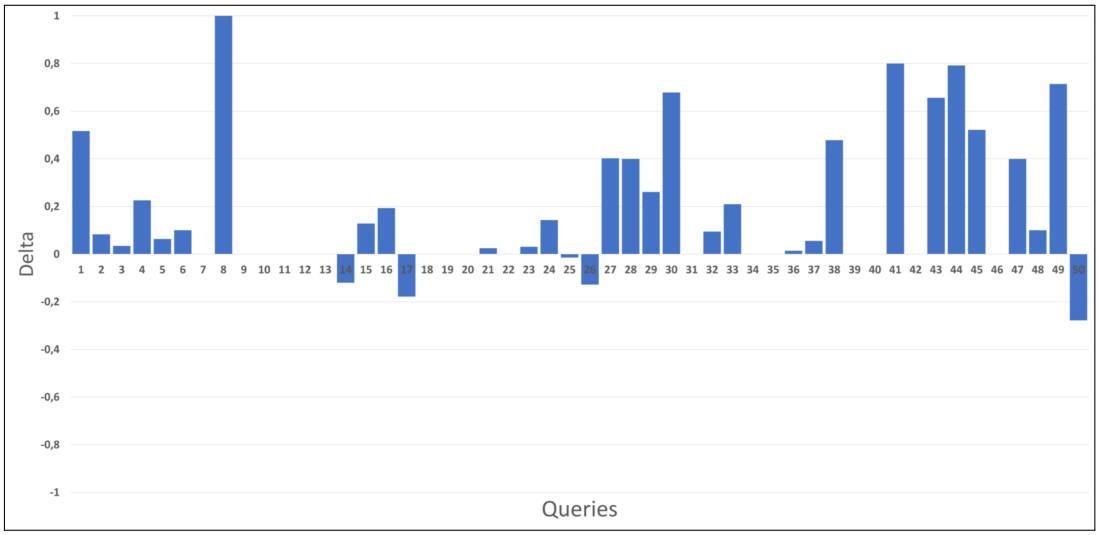

Figure 5.9: Differences in Recall between BruteForce and semi-TSA in IMDB 10% update using BM25 pipeline

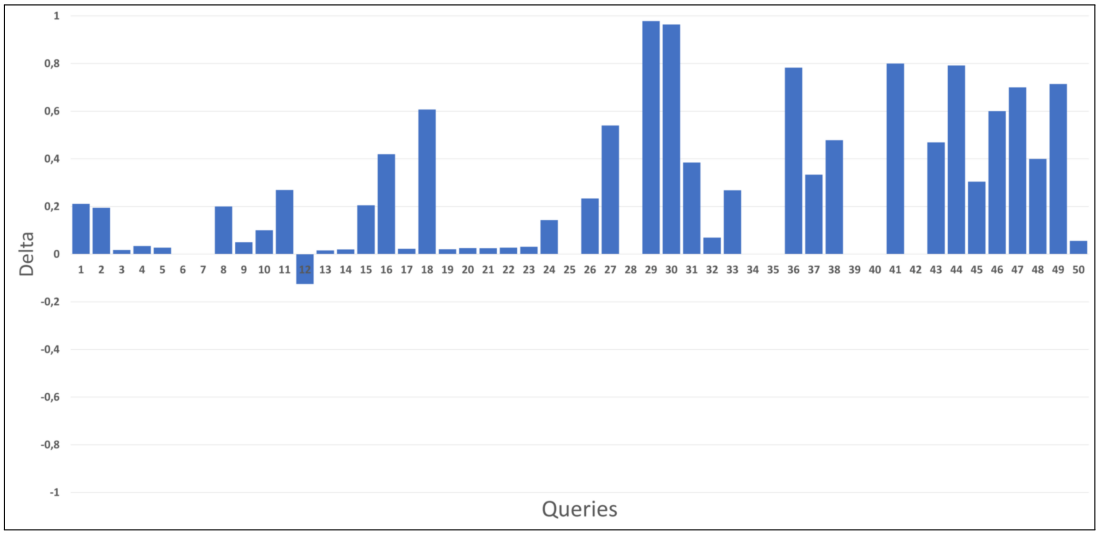

Figure 5.10: Differences in Recall between BruteForce and semi-TSA in IMDB 20% update using BM25 pipeline

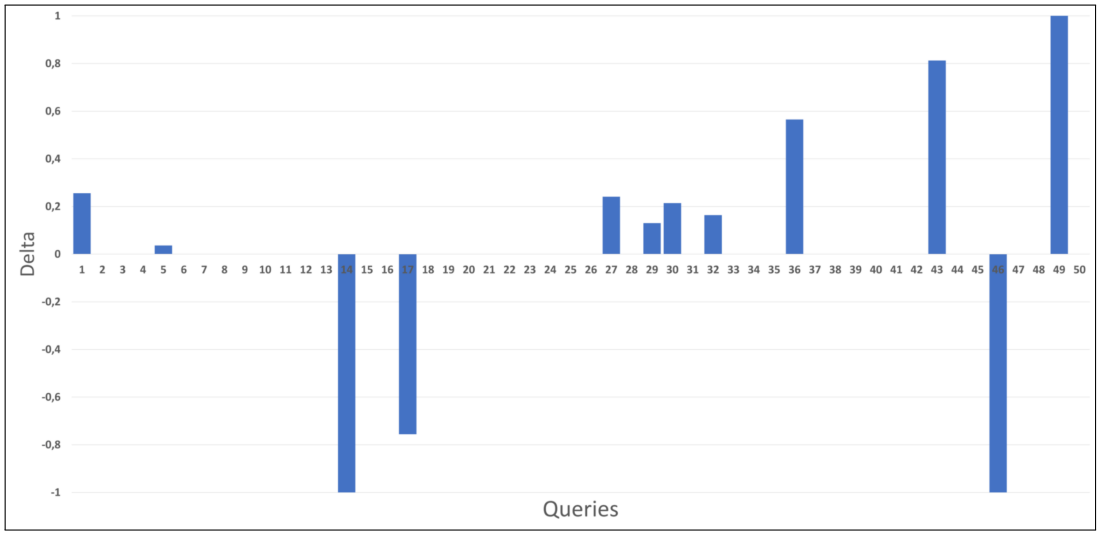

Figure 5.11: Differences in Recall between BruteForce and semi-TSA in IMDB 10% update using VDP pipeline

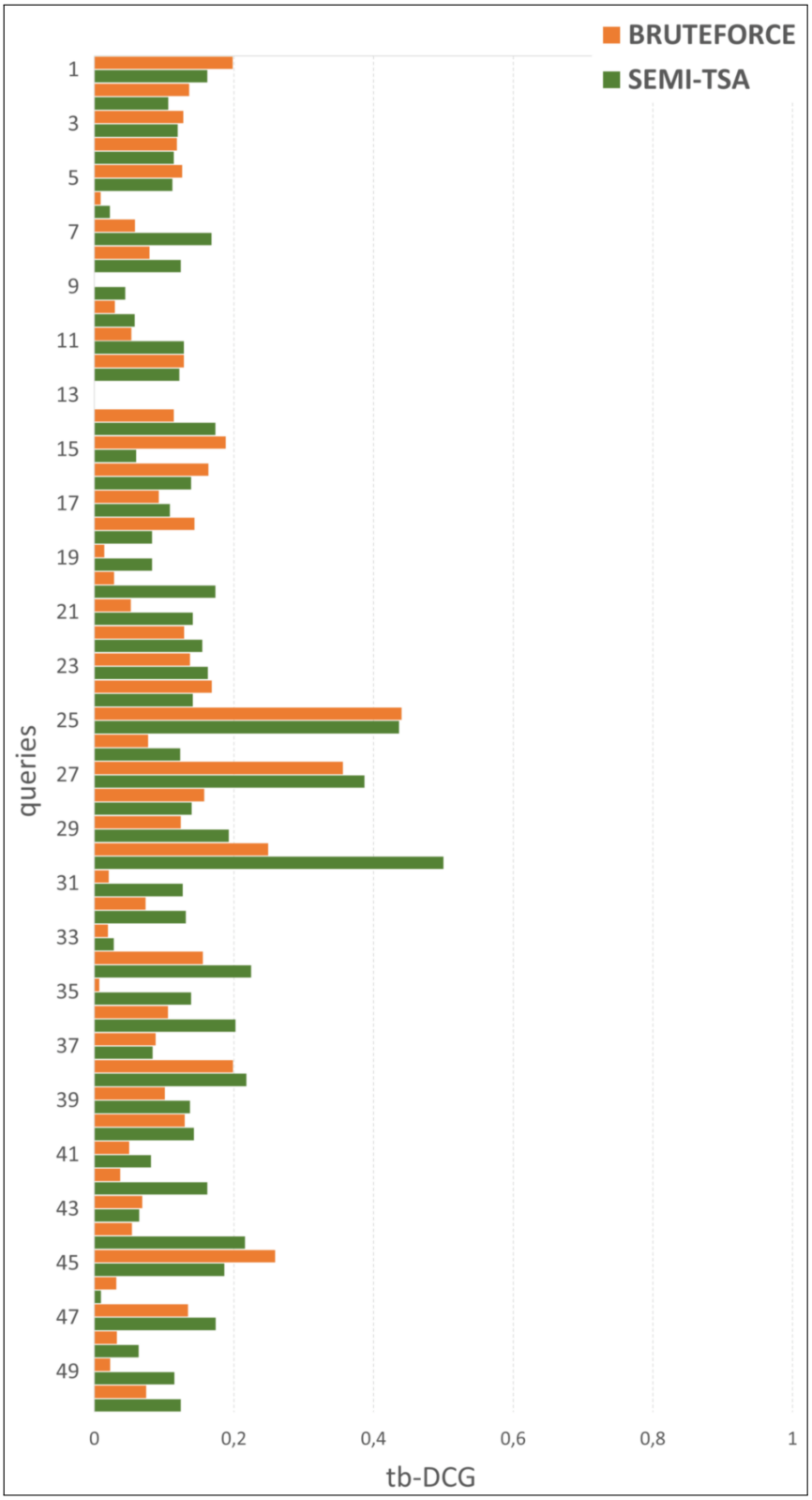

Figure 5.12: Comparison between methods using BM25 in IMDB tb-DCG at 10%

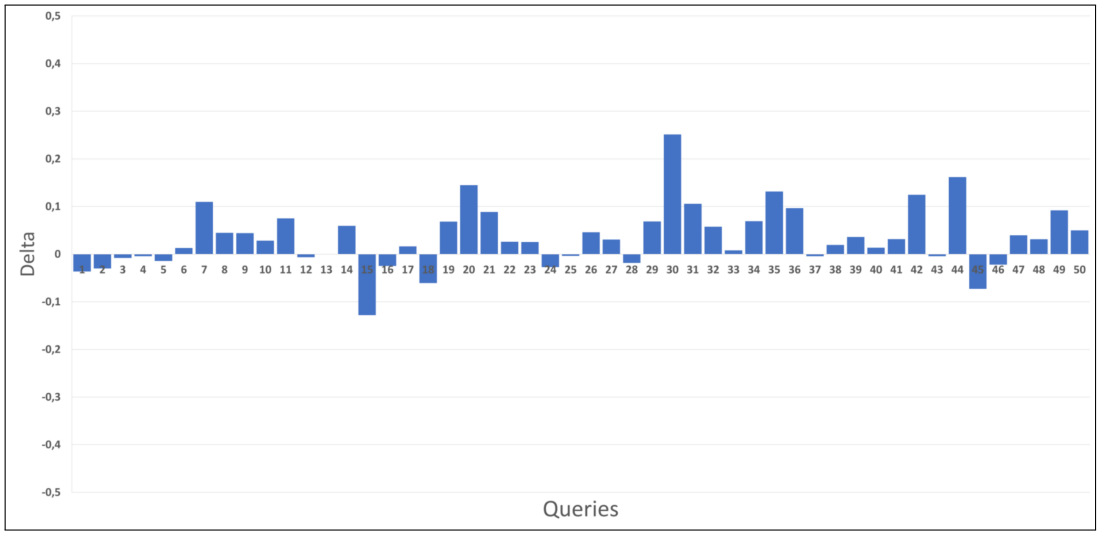

Figure 5.13: Differences in tb-DCG between BruteForce and semi-TSA in LinkedMDB 30% update using BM25 pipeline

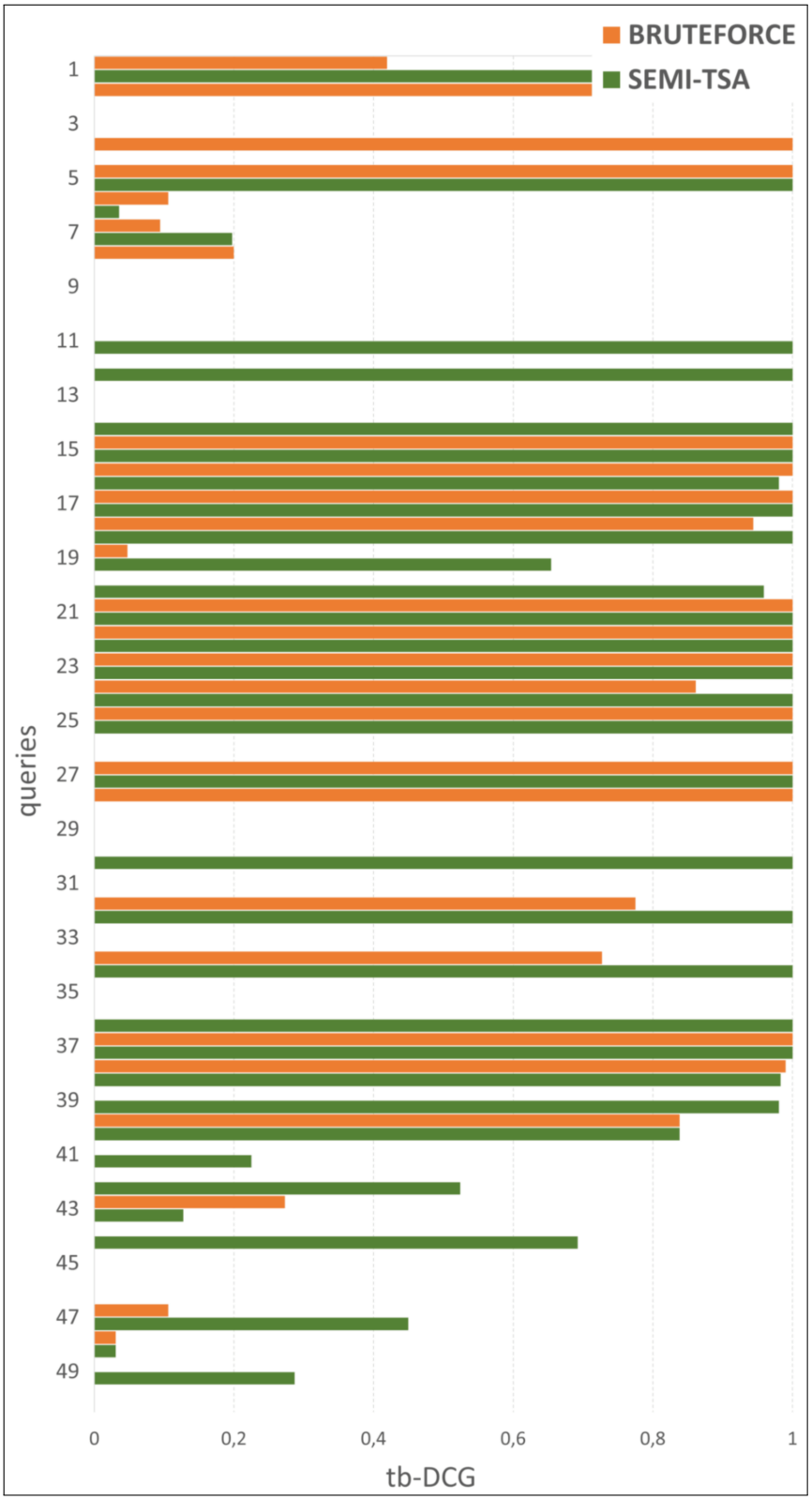

Figure 5.14: Comparison between methods using VDP in LinkedMDB tb-DCG at 30%

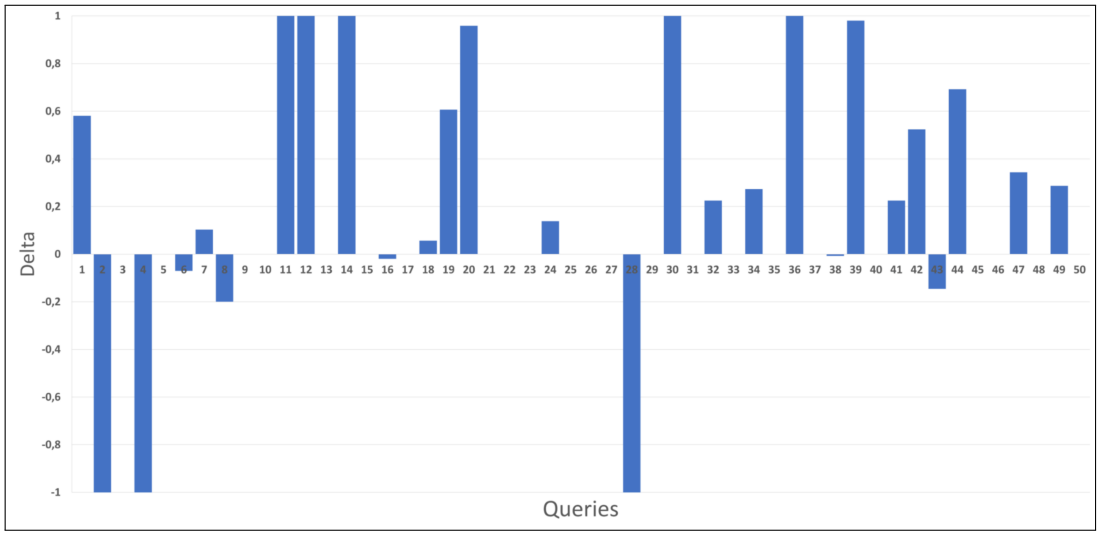

Figure 5.15: Differences in tb-DCG between BruteForce and semi-TSA in LinkedMDB 30% update using VDP pipeline

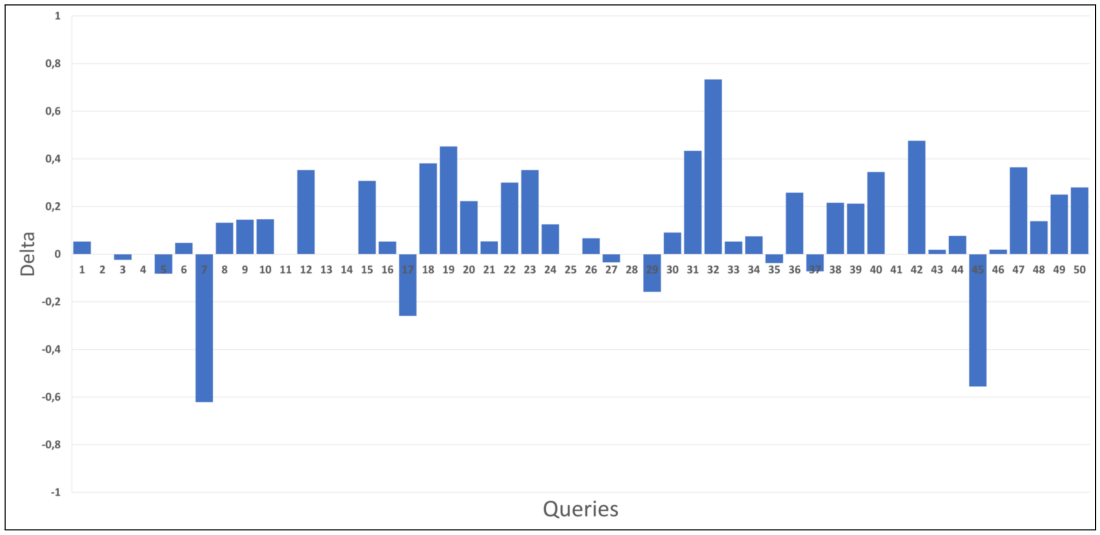

Figure 5.16: Differences in recall between BruteForce and semi-TSA in LinkedMDB 10% update using BM25 pipeline

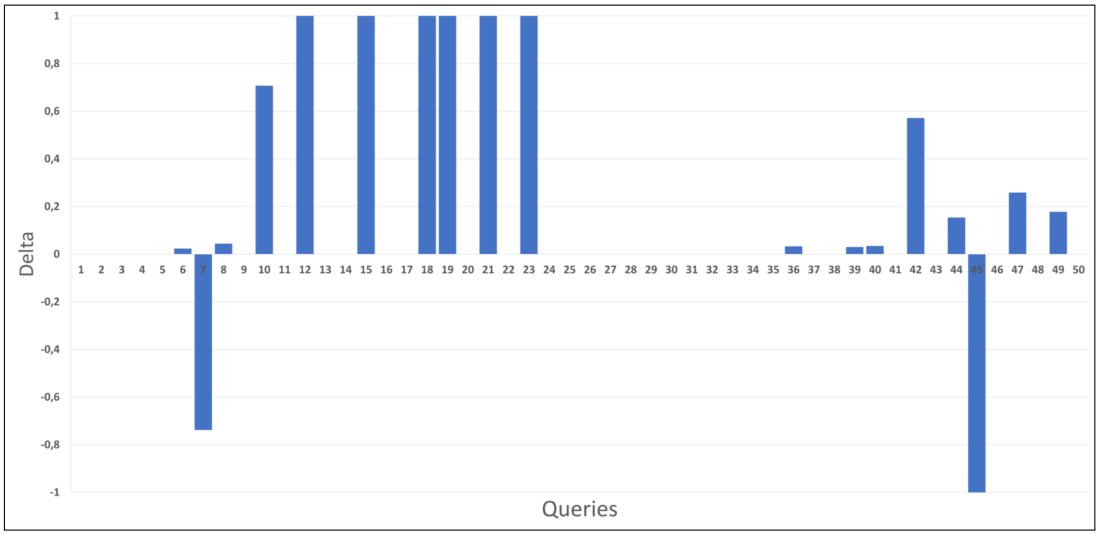

Figure 5.17: Differences in recall between BruteForce and semi-TSA in LinkedMDB 10% update using VDP pipeline

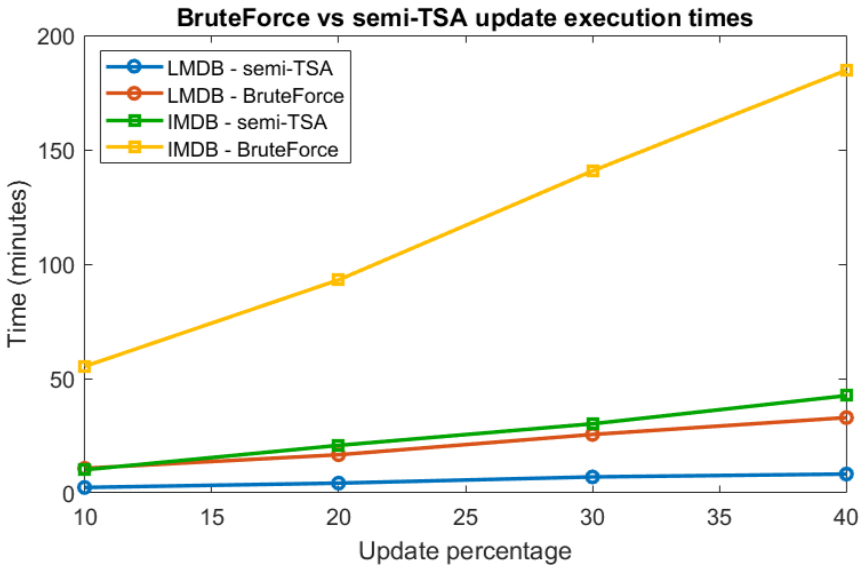

Figure 5.18: Processing time of the different update percentages between BruteForce and TSA

5.5 Multiple updates

Dataset	percentage of update	Systems	tb-DCG	recall	prec@1	prec@5	num of clusters
LinkedMDB	0 %	BM25	0.183	0.855	0.000	0.006	471.0 k
		VDP	0.537	0.861	0.034	0.032	
	10 %	BM25	0.126	0.583	0.002	0.008	514.0 k
		VDP	0.364	0.395	0.052	0.044	
	20 %	BM25	0.118	0.589	0.005	0.004	538.6 k
		VDP	0.437	0.467	0.020	0.019	
	30 %	BM25	0.140	0.518	0.001	0.002	561.2 k
		VDP	0.567	0.436	0.027	0.026	
	40 %	BM25	0.161	0.485	0.002	0.002	579.6 k
		VDP	0.534	0.413	0.031	0.029	

Table 5.5: Results obtained with update split in 5

Why not stress the system and see if there are any drops in the performance? This is the question we asked ourselves and tried to answer by dividing each update into five pieces to represent five different updates, performed one after the other. To be clear, an update of 30% has become 5 updates of roughly 360 thousand triples each. Previous experiments showed that the BruteForce system was not suitable for handling updates, so it was excluded. Mainly due to the high execution time and its overall low performance, such as tb-DCG or recall.

The experiments were conducted only on LinkedMDB because of its dimension. Table 5.5 shows that despite what we were expecting, so a degradation in performance, the results with LinkedMDB are quite good. There are even improvements over the single update, like in the 30% case. At 10% the number of new triples is 120 thousand, tb-DCG level is slightly worse than the single update, maybe this is connected to the fact that some triples are excluded and so split in more documents, which would not have happened having all the triples available at the same time. BM25 seems to perform fine in this situation as well. There is no noticeable difference with a whole update. This may be due to the characteristic of the database: the interconnections between the various clusters are clearly evident when considering the entire database, because tb-DCG is higher and the relevant triples are more similar to the ground truth, but when the single update is generated and triples are taken out then the few connections that were there between the subgraphs, such as director who directed multiple films or actor who participated in multiple films, are missing and are in such

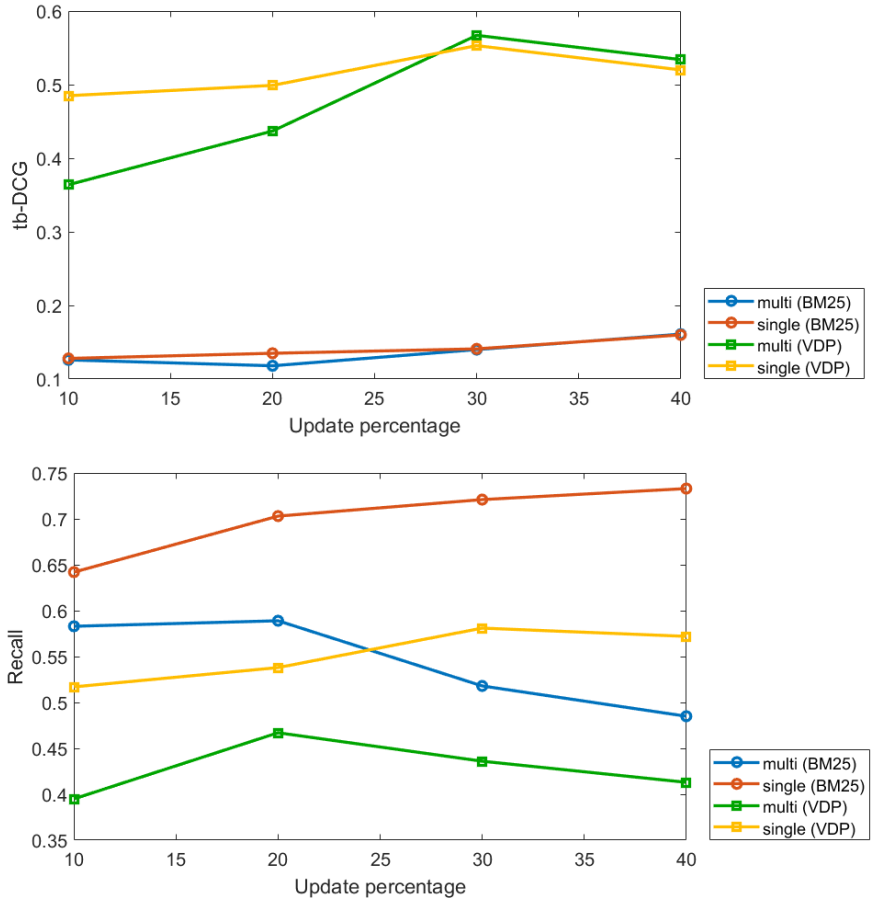

Figure 5.19: (a) tb-DCG results with multiple updates compared to a single one. *e.g. 20% means 5 updates of 4% each, and the other percentages follow the same pattern.* (b) Recall performance between a single and a multiple update.

numbers that the performance is lowered a bit, but not so much that the performance is worse in the case of multiple updates. TSA on the original dataset would have been able to capture this subgraph completely. Something else that perhaps helps in these results is the merging that occurs in the last steps of BM25 and is apparently useful.

Different story for VDP, which suffers from the inaccurate ranking of BM25 and the fact that it only works on the first 120 ranking results. Here the performance is lower with the smaller updates from 2% and 4%. In contrast, the performance is even better in the 6% and 8% cases. A larger update brings more triples, not only relevant but also noisy, which, however, VDP successfully eliminates by increasing the final SNR and yielding a better ranking.

Looking at recall, the differences are striking and clear in both pipelines,

around 20% less with respect to a single big update. In every update condition the single update generates a higher recall. As the rate of a single update increases both pipelines manage to increase the recall rate; with a fragmented update the recall drops visibly with BM25 approaching VDP levels, never before achieved in this dataset. The worst achievements are again with the small 2% updates(10%) like it was observed in the tb-DCG metric. The difference becomes important and marked by looking at the rightmost part of the two graphs, where with 5 updates with a size of 8 percent each the performance compared to a single update is truly remarkable. However, with the same percentage, the systems limit the damage and generate a very good ranking with the few retrieved documents.

6

Conclusions

The goals mentioned in the introduction of this paper have been achieved. We were able to reproduce the TSA+BM25 and TSA+VDP systems, in fact, we obtained results in line with those of the original paper. In some cases, the final results were better than the ones of the original paper due to a more accurate parameter setting. In this work, we also focused on developing two systems to manage updates in a keyword search over graph data system: BruteForce and semiTSA. The topic of updates on a keyword search system on an RDF graph had not yet been addressed in the literature, so we made our small contribution to open up a possible path for further research. An update was constructed by removing a certain amount of subgraphs from an RDF graph based on the size of the starting dataset. In order to evaluate the response of the two developed systems, different situations have been settled considering various update conditions. The changes to process updates were limited to the offline phase of the original TSA system, the online part remained the same as in the original paper, with both the BM25 and VDP pipelines handling the response to queries.

BruteForce has proven to be an inefficient and ineffective system compared to semiTSA. In all the performed tests on the two datasets considered, IMDB and LinkedMDB, BruteForce scored lower. In particularly tb-DCG metric was the one in which BruteForce did not perform well. tb-DCG is a measure that mimics the better-known DCG or nDCG metric in the IR landscape and expresses the system's ability to produce a good ranking. BruteForce also has the highest execution time and ends up analyzing the few triples in the update in a longer time than those in the base dataset. This fact, combined with the disappointing performance, suggests that the strategy adopted was not optimal.

On the other hand, semiTSA has shown to be the winning strategy among the two systems to handle a variable size update. Many times only semi-TSA was able to provide an answer to a query, and this suggests that improvements

could be made. In particular, it is evident that recall is the parameter that drops the most relative to the standard situation, where the entire dataset is available from the start. The two new systems cannot find all the triples useful to answer the user's query. This fact suggests that useful information was not perfectly reconstructed as in the baseline situation but remained separate, and only a portion of relevant triples were retrieved.

In the future, the update management approach could be improved by evaluating the use of separate parameters for creating subgraphs and the related documents between the base dataset and the update dataset. In addition, it remains to be addressed how to handle the cases of removal of triples or eventual spot changes, maybe relying on possible metadata.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. "DBXplorer: A system for keyword-based search over relational databases". In: *Proceedings 18th International Conference on Data Engineering*. IEEE. 2002, pp. 5–16.

[2] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, and Y. Velegrakis. "Combining user and database perspective for solving keyword queries over relational databases". In: *Information Systems* 55 (2016), pp. 1–19. DOI: https://doi.org/10.1016/j.is.2015.07.005. URL: https://www.sciencedirect.com/science/article/pii/S0306437915001337.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. "Keyword searching and browsing in databases using BANKS". In: *Proceedings 18th international conference on data engineering*. IEEE. 2002, pp. 431–440.

[4] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. "Finding top-k min-cost connected trees in databases". In: *2007 IEEE 23rd international conference on data engineering*. IEEE. 2007, pp. 836–845.

[5] D. Dosso and G. Silvello. "A Document-based RDF Keyword Search System: Query-by-Query Analysis". In: *SEBD*. 2020.

[6] D. Dosso and G. Silvello. "Search Text to Retrieve Graphs: A Scalable RDF Keyword-Based Search System". In: *IEEE Access* 8 (2020), pp. 14089–14111.

[7] D. Dosso and G. Silvello. "Virtual Document-based Methods for Keyword Search on RDF Graphs". In: *IIR*. 2019.

[8] D. Dosso and G. Silvello. "A Scalable Virtual Document-Based Keyword Search System for RDF Datasets". In: July 2019, pp. 965–968. ISBN: 978-1-4503-6172-9. DOI: 10.1145/3331184.3331284.

[9] S. Elbassuoni and R. Blanco. "Keyword Search over RDF Graphs". In: *Proc. 20th ACM CIKM*. New York, NY, USA: Association for Computing Machinery, 2011, pp. 237–242. DOI: 10.1145/2063576.2063615.

[10] L. Feddoul. "Semantics-driven Keyword Search over Knowledge Graphs." In: *DC@ ISWC*. 2020, pp. 17–24.

[11] S. Han, L. Zou, J. X. Yu, and D. Zhao. "Keyword Search on RDF Graphs - A Query Graph Assembly Approach". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. CIKM '17. Singapore, 2017, pp. 227–236. DOI: 10.1145/3132847.3132957. URL: https://doi.org/10.1145/3132847.3132957.

[12] O. Hassanzadeh and M. Consens. "Linked movie data base". In: (2009). URL: http://ceur-ws.org/Vol-538/ldow2009_paper12.pdf.

[13] V. Hristidis and Y. Papakonstantinou. "Discover: Keyword search in relational databases". In: *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier. 2002, pp. 670–681.

[14] Y. T. Izquierdo, G. M. Garca, E. Menendez, L. A. P. Leme, A. Neves, M. Lemos, A. C. Finamore, C. Oliveira, and M. A. Casanova. "Keyword search over schema-less RDF datasets by SPARQL query compilation". In: *Information Systems* 102 (2021), p. 101814.

[15] K. Järvelin and J. Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Trans. Inf. Syst.* 20 (2002), pp. 422–446.

[16] G. Kadilierakis, C. Nikas, P. Fafalios, P. Papadakos, and Y. Tzitzikas. "Elas4RDF: Multi-perspective triple-centered keyword search over RDF using elasticsearch". In: *European Semantic Web Conference*. Springer. 2020, pp. 122–128.

[17] M. Kargar, A. An, N. Cercone, P. Godfrey, J. Szlichta, and X. Yu. "Meanks: Meaningful keyword search in relational databases with complex schema". In: *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 2014.

[18] W. Le, F. Li, A. Kementsietsidis, and S. Duan. "Scalable keyword search on large RDF data". In: *IEEE Transactions on knowledge and data engineering* 26.11 (2014), pp. 2774–2788.

[19] G. Li, S. Ji, C. Li, J. Wang, and J. Feng. "Efficient fuzzy type-ahead search in TASTIER". In: *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*. 2010. DOI: 10.1109/ICDE.2010.5447804.

[20] X.-Q. Lin, Z.-M. Ma, and L. Yan. "RDF Keyword Search Using a Type-based Summary." In: *Journal of Information Science & Engineering* 34.2 (2018).

[21] Y. Luo, W. Wang, and X. Lin. "SPARK: A keyword search engine on relational databases". In: *2008 IEEE 24th International Conference on Data Engineering*. IEEE. 2008, pp. 1552–1555.

[22] Y. Mass and Y. Sagiv. "Virtual documents and answer priors in keyword search over data graphs". In: *Proc. Workshops EDBT/ICDT Joint Conf.* Vol. 1558. Bolzano, Italy: Association for Computing Machinery, 2016.

[23] V. dos Santos and S. Lifschitz. "A semantic search approach for hyper relational knowledge graphs". In: *Anais Estendidos do XXXVI Simpósio Brasileiro de Bancos de Dados*. SBC. 2021, pp. 106–112.

[24] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. "Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data". In: *2009 IEEE 25th International Conference on Data Engineering*. IEEE. 2009, pp. 405–416.

[25] L. Wangchao, L. Feifei, K. Anastasios, and D. Songyun. "Scalable Keyword Search on Large RDF Data". In: *IEEE Tran. on Knowl. Data Eng* 26 (2014), pp. 2774–2788.

[26] D. Whittenbury. *imdb designing a MySQL database and performing etl for imdb dataset using python*. 2020. URL: `https://dlwhittenbury.github.io/imdb-2-designing-a-mysql-database-and-performing-etl-for-imdb-dataset-using-python.html` (visited on 12/07/2022).

[27] Y. Yang, D. Agrawal, H. Jagadish, A. K. Tung, and S. Wu. "An efficient parallel keyword search engine on knowledge graphs". In: *2019 IEEE 35th international conference on data engineering (ICDE)*. IEEE. 2019, pp. 338–349.

[28] Z. Yu, X. Yu, Y. Chen, and K. Ma. "Distributed top-k keyword search over very large databases with MapReduce". In: *2016 IEEE International Congress on Big Data (BigData Congress)*. IEEE. 2016, pp. 349–352.

[29] G. Zenz, X. Zhou, E. Minack, W. Siberski, and W. Nejdl. "From keywords to semantic queriesIncremental query construction on the Semantic Web". In: *Journal of Web Semantics* 7.3 (2009), pp. 166–176.

[30] H. Zhang, B. Dong, B. Feng, and B. Wei. "A Keyword Query Approach Based on Community Structure of RDF Entity Graph". In: *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE. 2021, pp. 1143–1148.

[31] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, and D. Zhao. "Semantic SPARQL similarity search over RDF knowledge graphs". In: *Proceedings of the VLDB Endowment* 9.11 (2016), pp. 840–851.