Received December 6, 2019, accepted January 11, 2020, date of publication January 15, 2020, date of current version January 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966823

Search Text to Retrieve Graphs: A Scalable RDF Keyword-Based Search System

DENNIS DOSSO AND GIANMARIA SILVELLO, (Member, IEEE)
Department of Information Engineering, University of Padua, 35122 Padova, Italy

Corresponding author: Dennis Dosso (dennis.dosso@unipd.it)

This work was supported in part by the Computational Data Citation (CDC-STARS) project of the University of Padua, and in part by the ExaMode Project, as a part of the European Union Horizon 2020 Program under Grant 825292.

ABSTRACT Keyword-based access to structured data has been gaining traction both in research and industry as a means to facilitate access to information. In recent years, the research community and big data technology vendors have put much effort into developing new approaches for keyword search over structured data. Accessing these data through structured query languages, such as SQL or SPARQL, can be hard for end-users accustomed to Web-based search systems. To overcome this issue, keyword search in databases is becoming the technology of choice, although its efficiency and effectiveness problems still prevent a large scale diffusion. In this work, we focus on graph data, and we propose the *TSA+BM25* and the *TSA+VDP* keyword search systems over RDF datasets based on the "virtual documents" approach. This approach enables high scalability because it moves most of the computational complexity off-line and then exploits highly efficient text retrieval techniques and data structures to carry out the on-line phase. Nevertheless, text retrieval techniques scale well to large datasets but need to be adapted to the complexity of structured data. The new approaches we propose are more efficient and effective compared to state-of-the-art systems. In particular, we show that our systems scale to work with RDF datasets composed of hundreds of millions of triples and obtain competitive results in terms of effectiveness.

INDEX TERMS Information retrieval, database, keyword search, RDF datasets, search methods, triples (data structure).

I. INTRODUCTION

In the last decade, the Web of Data emerged as one of the principal means to access, share, and re-use structured data on the Web [1]. Linked Data, in particular, is the paradigm that realizes the Web of Data typically represented as *Resource Description Framework* (RDF) datasets (or RDF graphs). RDF is widely-used for data publishing, accessing, and sharing since it allows flexible manipulation, enrichment, and discovery of data as well as encouraging interoperability. In the past years, we have seen a significant increase in the number of knowledge bases published as large RDF graphs, such as DBpedia,[1] Wikidata,[2] and OpenPHACTS.[3] The use of RDF is growing even for the representation and management of enterprise data with heterogeneous and very large graphs, often containing millions of edges [2].

Efficient and effective techniques to retrieve and access these data are of paramount importance to allow end-users to discover, consult, re-use, and share these resources.

The standard way to retrieve data from RDF graphs is through the powerful, yet complex, SPARQL language. This language is not intuitive for end-users and is also very different from a natural expression of their information needs [3]. Moreover, it requires some knowledge of the underlying schema of the database. Keyword search is adopted to overcome these information access barriers since it is the easiest and quickest approach for searching information. It has been successfully applied to retrieve unstructured documents, e.g., by Web search engines [4]. Keyword-based methods have over time gained importance both in research fields and in industrial activities as a means to facilitate and make more natural the access to structured data. Over the past several years, the research community and big data technology

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

[1]https://wiki.dbpedia.org/
[2]https://www.wikidata.org/wiki/Wikidata:Main_Page
[3]https://www.openphacts.org/

vendors have put great effort into developing new approaches for keyword search over databases [4]–[6], but no research prototypes have so far transitioned from proof-of-concept implementations into fully deployed systems [7].

Two main challenges need to be addressed to improve current keyword-based search systems over structured data: *effectiveness* and *efficiency*. Effectiveness concerns the ability of the system to address the user information need by providing the best possible ranking of the retrieved candidate answers; whereas, efficiency concerns the time and memory required to provide such answers. To target effectiveness, we need to develop methods for ranking candidate answers – i.e., RDF subgraphs in our case – considering their structure and semantic relevance for a user's keyword query. A keyword query over an RDF graph may produce many results, so it is highly beneficial to present the user with a ranked list of candidate answers, where the top elements are the most relevant ones to satisfy the information need represented by the query [8]. To target efficiency, we need to develop systems that adequately scale, regarding query execution time and memory consumption, to real-world dataset sizes. Scalability is ''the most pressing challenge'' [2] that search systems over structured data have to face. This is of striking evidence in particular for graph data such as RDF.

Amongst the various keyword-based search methods, the so-called *virtual documents approach* is one of the most promising in terms of scalability [9], [10]. Following such approach, text documents are built starting from the content of nodes and edges of the graph – native if working with RDF or inferred if working with a relational database – and then indexed and retrieved using *Information Retrieval* (IR) techniques [9]. The systems based on this method move off-line part of the computations (e.g., documents creation) and exploits IR ranking techniques and data structures (e.g., inverted indexes) to speed up the search process on-line. IR techniques scale well to large datasets but need to be adapted to the intrinsic complexity of structured data.

A common aspect to all the systems based on the virtual documents approach is that they start from a definition of an exact answer: an answer for the keyword query Q is a *non-redundant* subtree T_x of the data graph, such that T_x contains all the keywords of Q. While containment means that each keyword appears in at least one node, non-redundancy means that every answer does not have a proper subtree containing all the keywords of the query.

Nevertheless, we deal with ambiguous information needs expressed with a few keywords. Therefore, we claim that a search system returning a ranking of *best answers* – as search engines do – rather than a single *exact answer* – as databases do – is better suited to capture the different angles of the information need potentially implied by the query. Hence, we compare the commonly adopted exact-match approach against the best-match approach we pursuit, and we show that we return accurate and non-redundant answers, while scaling to large data graphs that cannot be handled by exact-match based systems.

In this work, we propose two new keyword search systems over RDF graphs based on the virtual document approach and following the best-match search paradigm – i.e., the *Topological Syntactical Aggregator+BM25* (TSA+BM25) and *Topological Syntactical Aggregator+Virtual Documents Pruning* (TSA+VDP). TSA is the name of the document building strategy (that refers to how the document is built), which is common to the two systems. They differ in the on-line phase: how the document is ranked and mapped back to answer subgraphs. The off-line phase is based on the idea that an RDF triple is characterized by the concepts it contains, and triples describing similar concepts will cluster together. When working on text retrieval, this idea is postulated by the *clustering hypothesis*, on which even recent IR systems rely on [11]. TSA takes an RDF dataset as input and creates subgraphs around a single ''topic'' characterizing the subgraph semantics. Each subgraph is then mapped into a text document and indexed. The text documents and the related subgraphs are used for retrieval purposes during the on-line phase. TSA+BM25 is based on the classical retrieval function BM25, which ranks the virtual documents based on the user's keyword query. Once ranked, the graphs corresponding to the documents are returned to the user following the order given by BM25. Instead, the on-line phase of TSA+VDP starts from the ranking produced by TSA+BM25. It then aggregates and prunes them according to heuristics to reduce noise and increase specificity. It finally re-ranks them based on their relevance to the query. Compared to state-of-the-art systems, the difference consists in the way documents are built, and if and how the answer graphs are aggregated and pruned after the first ranking phase.

Another important aspect of keyword search is the evaluation of the systems. In the last years, the research community has concentrated the efforts on the evaluation of keyword search over relational databases [4] and an extensive evaluation has been conducted on the subjects. In particular, [7], [12] present a benchmark for the assessment of the system's efficiency and effectiveness when using models that work on relational databases. Unfortunately, there is no similar benchmark to evaluate keyword search systems over graph data.

The **objectives** of this work are:

- Modern state-of-the-art systems today are not able to scale to real-world-databases of tens and millions of triples. Our objective is the creation of new systems for Keyword Search on real-world RDF graphs.
- Many keyword search systems use a precise definition of the answer graph. We want our systems to be more focused on answering the information needs implied by the query rather than searching for particular subgraph structures inside the database.
- We aim to develop an evaluation framework designed for keyword search on RDF datasets based on a set of different RDF graphs, with different structures and sizes, and a set of keyword queries which can be easily reproduced and shared among researchers. Moreover,

traditional evaluation measures do not consider all the peculiar characteristics of the answer graphs, such as the different dimensions of the graphs, the overlapping among graphs, the presence of noise within the graphs. We aim at developing one new measure in order to take into consideration all of these elements in a holistic way.

Contributions of this paper include:

- Two new virtual document-based keyword search systems (TSA+BM25 and TSA+VDP) tested against three state-of-the-art baselines: the *Structured Language Model* (SLM) [13], *Markov Random Fields-Keyword Search* (MRF-KS) [14], and SUMM [15]. TSA+VDP shows statistically significant improvements over the baselines in terms of efficiency (up to one order of magnitude) and effectiveness. Moreover, TSA+BM25 and TSA+VDP are the only systems scaling to hundreds of millions of triples.
- A new evaluation framework based on the Cranfield paradigm, the standard for off-line evaluation of search engines working with the best-match approach. The framework consists of three real RDF datasets and two synthetic ones, 50 query topics per each real database, and their corresponding keyword query reformulations. For the synthetic databases, we used the topics provided released with the datasets. We construct the *Ground Truth* (GT) by leveraging on the answer graphs returned by the SPARQL queries that were manually built to match the information needs synthesized by the topics.
- An evaluation metric (i.e., tb-DCG) to measure the overall utility of a subgraph ranking for the end-users. It weights the top-heaviness (best answers are ranked first) and essentialness (absence of redundancy) in the ranking.

The rest of the paper is organized as follows: Section II, the related work. Section III, basic concepts about RDF, SPARQL and keyword search. Section IV, running experiment used as a reference throughout the paper. Section V, a detailed description of the functioning of the three baseline search systems we consider. Section VI describes TSA+BM25 and TSA+VDP. Section VIII, the Evaluation Setup of our experiments. Section VII, the design and use of the evaluation framework. Section IX, the experimental results. Finally, Section X draws some conclusions and discusses future work.

II. RELATED WORK

We classify the approaches to keyword search over structured data by query type, search paradigm, and data model.

Queries can be classified in *natural language queries* or *keyword queries*. Natural language queries are longer and have a stronger expressive power than keyword queries. Generally, they are considered by systems where the query is translated in a structured language (e.g., SQL or SPARQL) [16] via machine-translation techniques or computer-aided interfaces [17]. With shorter keyword queries it becomes harder to guess precisely the information need [4] and to "understand" the meaning inferred by the query. On the other hand, queries are more natural to formulate, and end-users are accustomed to using this paradigm for research thanks to services like Google and Amazon.

There are two main *search paradigms*: exact-match and best-match. Structured query languages work under the exact-match paradigm when a query is precise, unambiguous, and with only one correct answer. Within the best-match paradigm, the user query needs to be interpreted and inferred from a few keywords. The search system then returns a ranking of answers, ordered by relevance to the query. Ideally, the answers returned at the top of the ranking are more relevant than those at the bottom, and they are *essential*, i.e. without overlapping.

Keyword search systems have proved their flexibility in different applications and conditions, also thanks to supporting techniques such as QE [18]. They have been used, for example, to query knowledge bases [19], temporal graphs [20], and probabilistic graphs [21].

The search system we propose works with keyword queries, under the best-match assumption, and on RDF graphs. The task we face is challenging as we need to guess the information need of a query, and it is not straightforward to define what it means to be "relevant" for an answer [4].

In the literature, keyword query search systems over structured data are mainly focused on relational databases (RDB), leveraging on *schema-based* or *graph-based* techniques [6]. Schema-based techniques (e.g., DISCOVER [22], DBXplorer [23] and SPARK [24]) exploit the schema information to formulate SQL queries designed to start from the user keyword queries. Most of the existing approaches rely on indexes and functions over the data values to select the most prominent tuples as results of queries. [25], [26] are schema-based RDF search systems that translate keyword queries into SPARQL queries. Graph-based techniques (e.g., BANKS [27], PRECIS [28], DPBF [29] and STAR [30]) model relational databases as graphs, where the tuples are nodes and the foreign-primary key relations among them define the edges. The main difference within graph-based systems is how they optimize the computation of specific structures over the graphs (e.g., Steiner trees, rooted trees) to find the most relevant top-k connected tuples to be returned to the user. A core challenge is how to tackle the large graphs induced by the database instance. In several cases, the size of the graph makes it difficult to treat the problem.

All of these papers apply a particular definition of the answer to a keyword query, that is, the Tree Answer, derived from the one found in [6]. The Tree Answer is a tree-shaped answer graph where all the leaf nodes contain at least one keyword, and all of the query keywords are contained in the answer. In our work, we do not give a structural definition of an answer, but we look for a more user-oriented one. Our focus is not only on the presence of all the keywords inside the answer graph or in a particular structure but rather on its ability to cover the user's information need.

Keyword search systems usually employ some advanced heuristics to speed up the computations required to deal with such large and complex graph structures. Even so, in many cases, they cannot query a database within a reasonable time and according to memory limits, as shown by [7]. In the latter, most of the state-of-the-art RDB search systems were assessed against a shared test collection. Even though the evaluation effort attracted some criticism [31] (i.e., most of the queries consist of a single keyword, the datasets employed are modest in size, and the evaluation metrics are often inadequate to evaluate schema-based approaches), it remains the only shared testbed available to evaluate keyword search systems. Worth noting, the latter evaluates the effectiveness (quality of the ranking) and the efficiency (time and required memory) of the systems. This evaluation has shed light on the scalability problems that affect all the systems tested and on the query execution time, which grows with the number of keywords considered.

Most virtual document-based systems work for RDB and need to be revised to work with data graphs [24]. RDF graphs are very different from the graphs inferred from RDB. RDF nodes and edges are IRIs or literals, usually containing few keywords, while a node in an RDB graph may contain the text derived from the DB metadata and tuples. Hence, RDB graphs enable the creation of longer textual documents than those generated from RDF graphs. Moreover, the edges of RDB graphs are not labeled as in RDF, where they play a central role in searching.

Only a few search systems work on RDF graphs, or that can be easily adapted to work with them. The most relevant one is [14] – i.e., the *Markov Random Fields-Keyword Search* (MRF-KS) system – which leveraged on the "virtual documents" approach in searching data graphs. It also obtained the best effectiveness and efficiency performances against the state-of-the-art systems tested on the shared relational benchmark proposed by [7]. This approach leverages on the graph structure inferred from a relational database and adopts advanced IR techniques to index and rank. We use this system as a competing baseline for comparison with our systems. Since the MRF-KS system was originally designed for RDB, we adapted it to work with RDF, as described in Section V-B.

The *Structured Language Model* (SLM) system [13] is among the few native RDF search systems. It starts the searching process from the triples that match at least one of the query keywords and then produces answers by seeking connected components in the graph. SLM is based on virtual documents and employs IR models (statistical language models) for retrieval. SLM returns a ranking of RDF subgraphs to the user and not virtual text documents as in [9], [10].

SUMM [15] is a native RDF search system. The core idea behind this system is to produce a first partition of the whole graph through a BFS-based greedy algorithm. The partitions are edge-disjointed but are connected one to the other through nodes called portal nodes. Once given the keyword query, a backtracking algorithm similar to BANKS [27] is deployed to find connected subgraphs containing all the keywords. On these subgraphs, a different backtracking algorithm similar to BLINKS [10] is used to produce tree-shaped answer graphs whose leaves are nodes containing one keyword each. The systems use indexes and particular stopping conditions to speed-up the execution.

Virtual document-based systems have the potential to scale more than schema- and graph-based systems, but they are often limited by the graph exploration strategies they employ that do not scale effectively when the database size increases. In this paper, we discuss the limitations of current keyword search systems over RDF and analyze why they cannot scale to real-world datasets.

III. PRELIMINARIES

In this section, we present some basics on RDF, SPARQL and keyword search. For a more detailed description on RDF-related concepts, refer to [32]–[34].

The Resource Description Framework (**RDF**) is a family of specifications developed and supported by the W3C[4] to represent information on the Web. In particular, RDF is mainly used to publish and interlink data on the Web, allowing a definition of *statement* about any kind of resource. RDF statements have a `<subject, predicate, object>` triple-based structure. In a triple, `subject` and `object` are resources, whereas the `predicate` is a string defining the *type* of relationship.

RDF resources are classified into *IRIs*(Internationalized Resource Identifier), *literals* and *blank nodes*. An IRI[5] is a string used to uniquely identify a resource on the Web (it is a generalization of URIs which can also contain UNICODE characters). In the RDF syntax, an IRI can either be a `subject`, an `object` or a `predicate`. A literal is a basic value associated with a data type, e.g. string, boolean, integer, and date. When a data type is not specified, the default is "string". Literals can only be `object`s of a triple. A blank node is a resource without a global identifier. A blank node can be a `subject` or an `object` of a triple.

Definition 1: Let I, B and L be three pairwise disjoint sets of IRIs, blank nodes and literals, respectively. We define $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ to be an *RDF triple*. An RDF dataset $\mathcal{G} = \{I, B, L\}$ is defined as a set of RDF triples.

We can represent an RDF dataset as an edge-labeled directed graph where the subjects and objects are the nodes, and the edges are the predicates; the direction of the edges is always from the subject to the object. Hence, an RDF dataset can also be called RDF graph.

SPARQL[6] is a structured language for querying RDF datasets that allows the retrieval and processing of triples.

Definition 2: Let V be a set of query variables, disjoint from I, B and L, and $(I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V)$ be the set of *triple patterns*. We define a *SPARQL query* to be

[4]https://www.w3.org/TR/rdf11-primer
[5]https://tools.ietf.org/html/rfc3987
[6]https://www.w3.org/TR/sparql11-overview

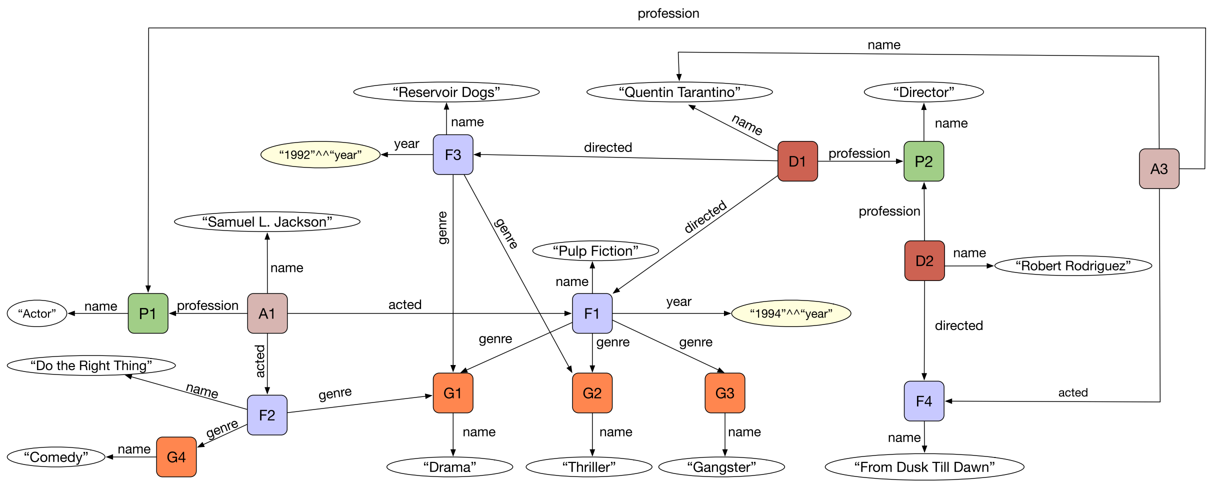

FIGURE 1. A sample RDF graphs derived from LinkedMDB.

a tuple of the form:

$$(\textit{query-type}, \textit{pattern } P, \textit{solution-modifier}).$$

The *pattern P* contains the triple patterns that have to be matched onto the RDF data. It is the syntactic part of a SPARQL query because it searches for specific subgraphs in the input RDF dataset. Its result is a (multi-)set of mappings (M) that match the pattern to the data. The actual output of a query is determined by the *query-type* that is one of four types: (i) `select` that returns projections of mappings from M; (ii) `construct` that returns a new set of triples based on the mapping in M; (iii) `ask` that returns `true` if the pattern P is matched in the input dataset or `false` otherwise; and, (iv) `describe` that returns a set of triples that represent the IRIs and blank nodes found in M. Finally, the *solution-modifier* allows aggregation, grouping, sorting and duplicate removal.

Definition 3: The SPARQL queries that have in their pattern P only triple patterns and contain only the AND operator (in SPARQL written with a dot) are called *conjunctive queries*. They are generated by the following grammar:

$$P \quad ::= \quad tp \mid P_1 \; AND \; P_2$$

where *tp* is a triple pattern.

A SPARQL conjunctive query presents a pattern P composed of triple patterns each connected to create an RDF graph. This graph is then matched against the data to retrieve the triple sets satisfying structure and content of the pattern.

In subsequent parts of this paper, we only consider SPARQL conjunctive queries with `construct` as query type. The query returns as answer a subgraph taken from the dataset which results from the matching pattern P to the dataset structure. This graph is called *answer graph*. One example of query that we used can be found in Example 1.

Keyword search is the foremost approach for information search which is being studied extensively in the IR field [35]. Retrieving information from documents is intrinsically different from querying databases, which are typically accessed through structured queries. A key difference is that the result set of a keyword search system is not "exact". In fact, the system performs queries, knowing that they may not precisely correspond to the user's information need. It must also disambiguate search terms and rank the results of the search considering the relevance for the user [31]. A typical keyword-based search process starts by considering the user's information needs. From this point, a query composed by one or more keywords is derived and submitted to a search system which returns a list of results (e.g., documents for unstructured search, tuples for relational DBs or set of triples for RDF datasets) ranked accordingly to their estimated relevance for the user. The ordered list of results returned is what we call *ranking*.

IV. RUNNING EXAMPLE

Let us consider a simple RDF graph (Figure 1) derived from the LinkedMDB dataset.[7] The main classes of entities are film (`F`), director (`D`), actor (`A`), profession (`P`) and genre (`G`). In a real RDF graphs the entities belonging to these classes are identified by their IRI or by a `type` predicate. In the example, these entities are identified by a simple synthetic IRI, like A1 for the actor "Samuel L. Jackson" or D2 for the director "Robert Rodriguez." A directed and labeled edge connects each pair of nodes.

Example 1: Given the RDF graph in Figure 1, a possible information need is: *The title of the films directed by Quentin Tarantino*. A conjunctive SPARQL query Q_s, producing the RDF graph of the exact answer shown in Figure 2, is:

```
CONSTRUCT
WHERE {
?director_iri <name> <‘‘Quentin Tarantino’’>.
?director_id <directed> ?film_iri.
?film_id <name> ?film_name. };
```

The pattern of the query matches five triples, identifying the director "Quentin Tarantino" with its IRI `D1` and the

[7]http://www.linkedmdb.org

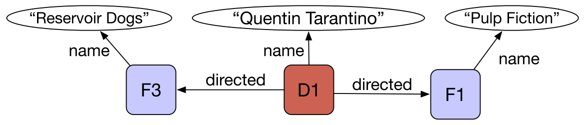

FIGURE 2. Exact answer to the SPARQL query reported in Example 1.

two films (F1 and F3). All the triples in the answer graph are relevant to the user, and there are no noisy (i.e., not-relevant) triples. Note that the query uses the `construct` type because we want the answers of the system to be graphs.

As we can see, writing a SPARQL query is not straightforward: it requires knowing the syntax of the language, the structure of the dataset, and also the IRIs used inside the graphs to correctly formulate the pattern P. On the other hand, to query this graph with a keyword search system is straightforward, but the system needs to: (i) match the query keywords to the entities of the knowledge base; (ii) generate the connected subgraphs that include the matching entities by browsing the graph and also adding triples to the answer that do not necessarily match any keyword; and (iii) rank them by relevance to the user information need [4].

Example 2: Let us consider the SPARQL query Q_s in Example 1, that we can reformulate as a keyword query Q_k, such as: "Films directed by Quentin Tarantino." The "exact" answer graph for this query is shown in Figure 2.

Nevertheless, keyword search systems work under the best match assumption and return a ranking of candidate answers. Hence, a possible output from a hypothetical system is shown in Figure 3, where there is a ranking of three answer graphs ordered by relevance to the query: A, B and C. We can see that no answer matches the exact one in Figure 2. In particular, the answer graph A contains all of the required triples, but it adds two additional triples (`<D1, profession, P2>` and `<D1, year, 1994>`) that are not strictly relevant and can thus be labeled as *noise*. The answer graph B is only partially relevant, given that it contains only a subset of the exact answer. Moreover, the triples in B are also returned by A. This means that a user, looking at this ranking, will find no additional relevant data in B compared to A. In this case, we say that B is not *essential* since it contains redundant triples.

The answer graph C is not relevant since it is about a film directed by "Robert Rodriguez" and not by "Quentin Tarantino." In this case, the system returned this answer because the subgraph matches all the query keywords, and the system could not differentiate the actor "Quentin Tarantino" from the director.

V. BASELINE KEYWORD-BASED SEARCH SYSTEMS

Keyword search systems based on the virtual documents approach take an RDF graph $\mathcal{G}$ and a user keyword query Q_k as input. They produce a set of textual documents $\mathcal{D}_G$, combining the strings extracted from IRIs and literals. They then apply a retrieval model on $\mathcal{D}_G$, returning a set of answer documents. They transform them into answer graphs, rank them according to their *relevance* to Q_k and return the final ranking to the user.

We define a general framework for virtual document-based keyword search system composed of four main modules, as shown in Figure 4:

1) *Subgraph Extraction Module*: creates a set of subgraphs from $\mathcal{G}$ using a custom graph extracting function that may exploit some heuristics (e.g., keyword matching, topic modeling, clustering).
2) *Data Extraction Module*: extracts support data from $\mathcal{G}$ such as the textual content of the nodes, their in-degree and out-degree, and the structure of nodes neighborhood.
3) *Document Creation Module*: creates the textual (virtual) documents by using the RDF subgraphs and the support data. Generally, a unique text document is created from a single RDF subgraph by linking the strings stripped from their IRIs and literals.
4) *Matching and Ranking Module*: retrieves the virtual documents relevant to the query, transforms them back into RDF graphs, merges/prunes/separates the graphs, and returns the ranking of answer graphs to the user.

The main differences between the proposed keyword search system and those in the literature are centered on all of the modules depicted in Figure 4 except the document creation module, which stays the same for all the keyword search systems we considered.

Another critical aspect differentiating keyword search systems, one from the other, is when they take into account the user query. Figure 4 illustrates the way a user query can be used starting from the subgraph extraction, the document creation, or the matching and ranking module. The use of the query determines when the on-line phase begins – the *response time* perceived by the user – and when the off-line phase ends. SLM performs most of the work on-line, while MRF-KS and SUMM balance the two phases. In the next sections, we describe the baseline systems functioning according to the presented framework in order to ease their comparison.

A. SLM

SLM is a native RDF keyword search system based on virtual documents proposed in [13]. The example presented in Figure 5 describes the functioning of SLM.

The SLM on-line phase consists in the creation of one virtual document per triple in the graph $\mathcal{G}$ and in the indexing of these documents. Thus it is generally completed in a short time.

The SLM on-line phase takes as input the index of the triples virtual documents and a keyword query Q_k. It then creates a single RDF subgraph called "Query Graph" E composed of all of the triples in $\mathcal{G}$, matching at least one keyword in Q_k. Without loss of generality, let us consider Q_k ="Films directed by Quentin Tarantino". Given the graph $\mathcal{G}$ shown in Figure 1, the matching words are "directed" and "Quentin Tarantino."

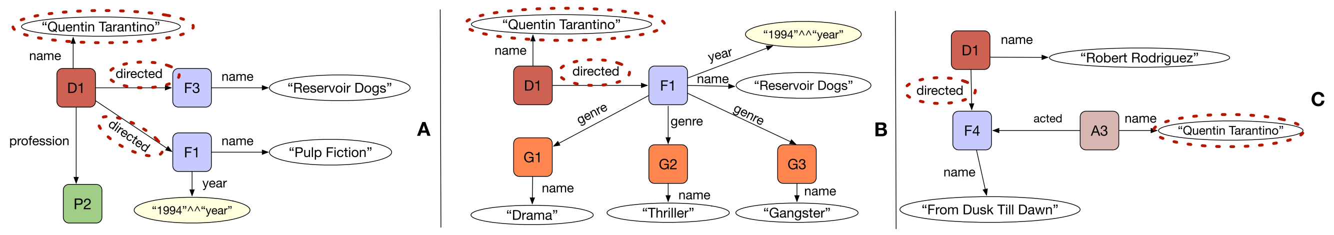

FIGURE 3. Ranking of answer graphs generated starting from the keyword query *Films directed by Quentin Tarantino*. The words matching the query keywords are circled in red.

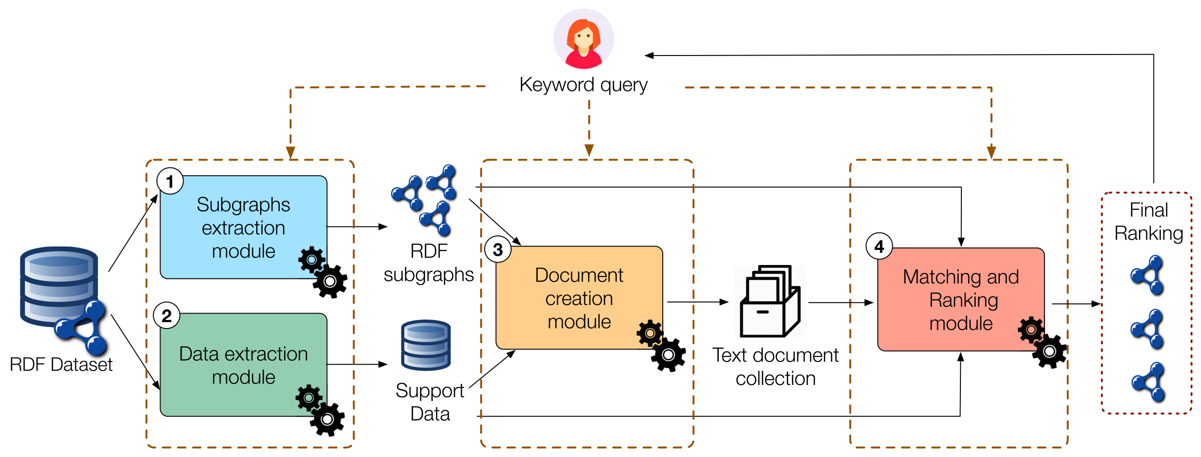

FIGURE 4. Overview of a general keyword search system based on the "virtual documents" approach.

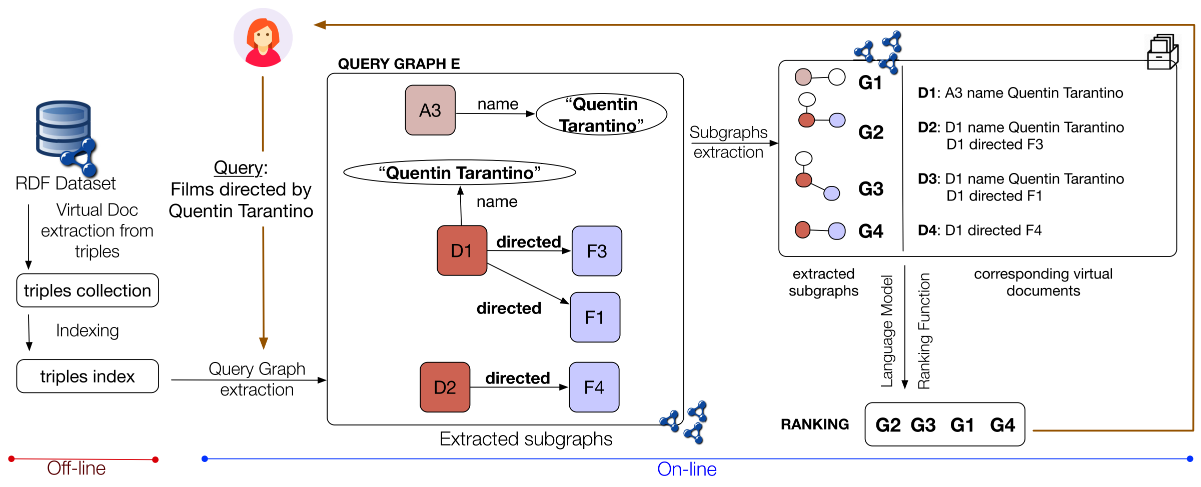

FIGURE 5. Example of how the SLM system works, based on the RDF graph shown in Figure 1.

The "subgraphs extraction" module extracts from $\mathcal{G}$ all the triples containing at least one of the query words. In our example, E is composed of three disconnected components, as shown in Figure 5.

Given the query graph E, the "subgraphs extraction" module builds the answer graphs that will be ranked by the ranking function. These subgraphs need to satisfy the following two properties: (i) *uniqueness* and *maximality*: every subgraph retrieved should not be a subset of any other subgraph retrieved; (ii) *diversity*: the subgraphs should contain triples matching different sets of keywords – that is, if two triples match the same set of keywords, they must be part of two different answers. Finally, one text document is derived from each one of the answer subgraphs.

Text documents are used as proxies to determine the similarity between the answer subgraphs and the query. The

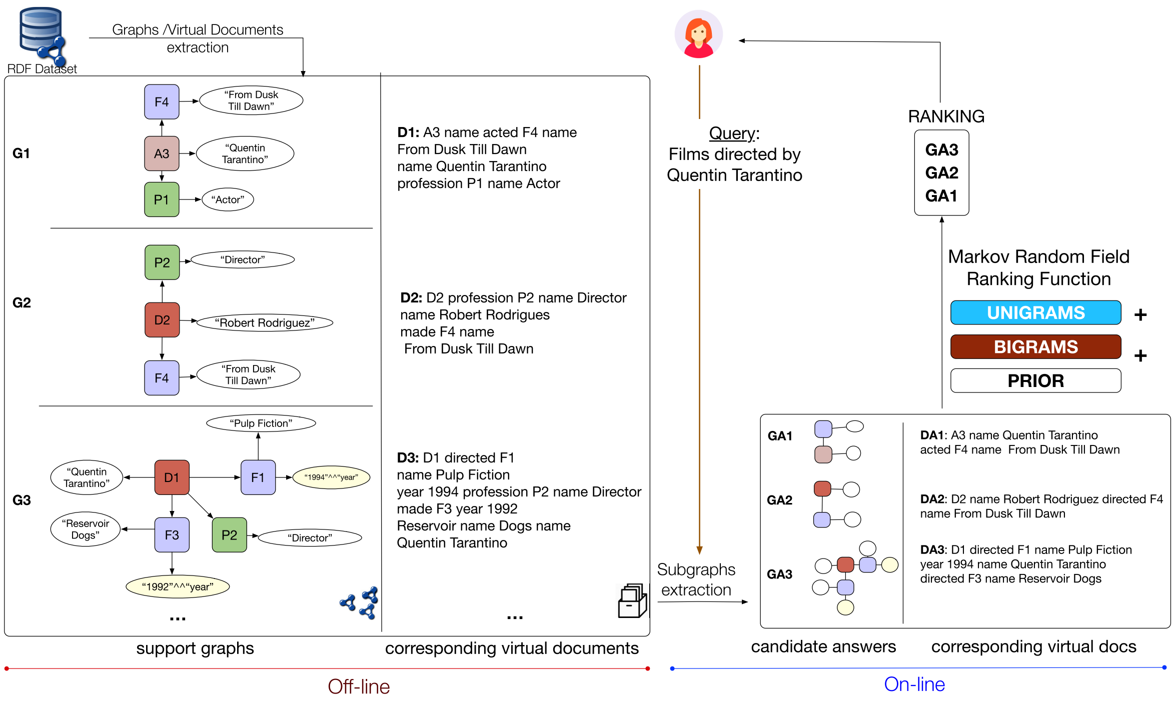
RDF Dataset
Graphs /Virtual Documents extraction
G1
F4
"From Dusk Till Dawn"
A3
"Quentin Tarantino"
P1
"Actor"
D1: A3 name acted F4 name From Dusk Till Dawn name Quentin Tarantino profession P1 name Actor
G2
P2
"Director"
D2
"Robert Rodriguez"
F4
"From Dusk Till Dawn"
D2: D2 profession P2 name Director name Robert Rodrigues made F4 name From Dusk Till Dawn
G3
"Pulp Fiction"
"Quentin Tarantino"
D1
F1
"1994"^^"year"
"Reservoir Dogs"
F3
P2
"Director"
"1992"^^"year"
D3: D1 directed F1 name Pulp Fiction year 1994 profession P2 name Director made F3 year 1992 Reservoir name Dogs name Quentin Tarantino
...
support graphs
corresponding virtual documents
Off-line
Query: Films directed by Quentin Tarantino
RANKING
GA3
GA2
GA1
Markov Random Field Ranking Function
UNIGRAMS +
BIGRAMS +
PRIOR
Subgraphs extraction
GA1
DA1: A3 name Quentin Tarantino acted F4 name From Dusk Till Dawn
GA2
DA2: D2 name Robert Rodriguez directed F4 name From Dusk Till Dawn
GA3
DA3: D1 directed F1 name Pulp Fiction year 1994 name Quentin Tarantino directed F3 name Reservoir Dogs
candidate answers
corresponding virtual docs
On-line

FIGURE 6. **Example of how the MRF-KS system works, based on the RDF graph shown in Figure 1.**

document-query similarity is determined using a ranking function based on a language model (LM). An LM ranks the documents using the query likelihood or the probability of generating the query Q_k given a document D. In this case, the system cannot use a standard LM since it would merely consider the frequency of the words within a text document. Hence, the graph structure to solve possible ambiguities present in the document must also be considered, since the probability of a term belonging to a triple does not only depend on the nodes, but also on the predicate connecting them. In our example, G2 and G3 are ranked before G1 because they both contain the predicates "name" and "directed". This is considered shared evidence that G2 and G3 are about a director name, and not about an actor name as was the case for G1.

The RDF-based LM defined for SLM considers the frequency of a predicate within a graph along with the subjects and objects that it usually connects. Thus, SLM weights the similarity of a graph with the query by considering both text and graph-based syntactical properties.

B. MRF-KS

MRF-KS was introduced in [14] to perform keyword search on data graphs. It was tested on the shared evaluation collection reported in [12] and achieved the best performances amongst all the state-of-the-art systems for keyword search on RDB.

We use the example shown in Figure 6 to describe the way it works. Given a node $v \in \mathcal{G}$, we call the *support graph* of v the graph composed of the nodes connected to v by a path with length $\tau \in \mathbb{N}^+$, where τ is a tunable parameter of the system. The aim of the MRF-KS off-line phase is to extract a set of *support graphs*, one for each node in RDF dataset $\mathcal{G}$, and their corresponding virtual documents (graphs G1, G2 and G3 in the Figure). In this phase, the system also retrieves the nodes' degree, their text content, and the neighborhood.

In Figure 6, we show only three of the support graphs created from the sample graph shown in Figure 1. In particular, the support graph G1 is created starting from node A3, G2 from node D2, and G3 from node D1, with $\tau = 1$.

The "subgraphs extraction module" takes the support graphs, the associated text documents, and the user query as inputs. Then, it selects the support graphs containing all the keywords and prunes them by keeping only the nodes within the paths between the root node v and a terminal node containing at least a query keyword. Hence, the answer graphs produced by MRF-KS are trees. In our example, the answer graphs GA1, GA2, GA3 are generated from G1, G2 and G3, respectively, following this strategy.

To rank the candidate answer graphs, MRF-KS employs a Markov random field model [36]. Built from a given graph, this model considers the nodes as random variables, while it uses the edges to define the independence semantics between them. It enables the incorporation of arbitrary text features as evidence. In particular, it considers the occurrences of unigrams (single words) and unordered bigrams (word pairs). It also incorporates a query-independent feature, i.e., the prior of an answer graph. This prior is computed from the multiplication of factors. Each factor is the ratio between the node degree and the sum of the nodes' degrees in the parent node

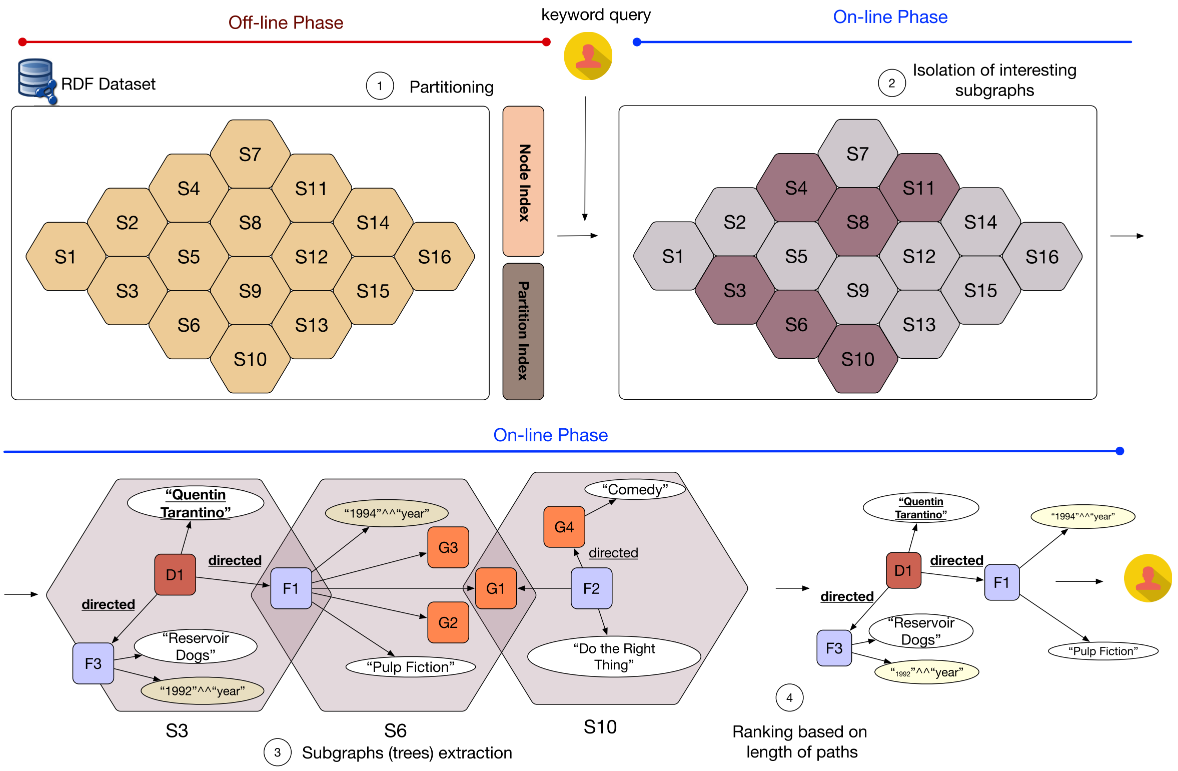

FIGURE 7. Functioning of the SUMM system based on the RDF graph shown in Figure 1 with keyword query "Films directed Quentin Tarantino".

neighborhood. In this sense, the prior considers the structure of the answer graph. In particular, the prior function gives higher scores to smaller trees awarding precision over recall.

In our example, the graphs GA3 is ranked at the top thanks to the presence of more query keywords (both grouped as unigrams and unordered bigrams) within shorter paths from the root.

C. SUMM

SUMM is a native RDF search system presented in [15] which scales over large RDF datasets. An example of its functioning is shown in Figure 7.

Given a keyword search query $q = \{w_1, \dots, w_m\}$ against an RDF dataset $\mathcal{G} = \{V, E\}$, a candidate answer to the query is a tree T induced by the set of vertices $t = \{r, l_1, \dots, l_m\} \in V$, where r is the root of the tree and a node l_i is a leaf of the tree. The set t is called *qualified candidate* to the query. A qualified candidate has to satisfy these properties: (i) the root answer node r is reachable by l_i $\forall i \in [1, m]$; and (ii) the vertex l_i contains the keyword w_i ($i \in [1, m]$). In our implementation one leaf node can contain one or more keywords. One qualified candidate can generate, through different paths, one or more candidate answers.

Let $C(q)$ be the set of all the qualified answers for a query q. The answer to q is given by: $\mathcal{A}(q) = \arg\min_{T \in C(q)} s(T)$, where $s(T) = \sum_{l_i \in T, i=1,\dots,m} d(r, v_i)$ and $d(r, v_i)$ is the distance between the root r and a leaf node v_i in the tree T. That is, the answer of a keyword query is the candidate answer tree of minimum weight, where the weight is computed by the sum of distances between the root and the leaves. A top-k version of this problem is about the retrieval of the top-k candidate from $C(q)$, that is, the first k answer trees with minimum weight.

An IRI node is considered together with all the literals surrounding it and the object nodes connected through the `type` predicate: this forms what we call a "meta-node".

During the off-line phase (phase 1, Figure 7), the algorithm divides the graph in an edge-disjointed partition using a BFS-based (*Breadth First Search*) algorithm. The nodes that belong to two or more partitions are called *portal nodes*. An *index of the words* in every node in the partitions is kept, together with an *index of the partitions* keeping track of the portal nodes connecting the different subgraphs. This index is used later in the on-line phase to navigate from one partition to the other.

In the first part of the on-line phase (phase 2, Figure 7), the algorithm seeks all the partitions containing at least one query keyword. All these partitions are the starting point of a backtracking algorithm that aims to find a subgraph composed of partitions containing all the keywords. In this backtracking, the portal nodes are the gateways from one partition to the other. The aim is to find groups of partitions connected among them, which contain all the keywords. Together, the partitions form a connected subgraph of $\mathcal{G}$. Once one such subgraph is found, another backward algorithm, inspired to BLINKS [10], is applied in order to build a candidate answer. In Phase 3 of Figure 7, we see how keywords contained in the subgraphs are used to build an answer tree (nodes F1 and G1 are portal nodes).

A stopping condition is used with the help of different data structures (a list of min heaps) to stop the execution without exploring all the space of possible answers. When the estimated weight of the tree (which still needs to contain all of the keywords) with the minimum weight is bigger than the weight of the k-th tree in the ranking produced so far, the stopping condition is met since new trees will always have bigger weight and will not end up in the top-k positions.

In the end, the algorithm produces k (or less) candidate weighted answer trees. The ranking function of SUMM simply orders the trees in ascending order of weight (phase 4, Figure 7), where the weight is computed summing the lengths of all the paths in the tree.

SUMM scales over big databases thanks to the design of its off-line phase, which is in part based on the virtual document strategy. Moreover, the use of a first backward phase over the partitions instead of over the single nodes resolves the limitations of previous algorithms, e.g. BANKS and BLINKS, where the backtracking strategy was instead applied directly to every node.

Note that since SUMM provides a definition of what an exact answer is, the analysis of the algorithm presented in [15] only covers the efficiency aspect of the system. SUMM, however, has some drawbacks for the end-users. In particular, since the system takes into consideration the top-k qualified candidate trees, these trees often differ in only one branch that leads to a different node containing the same keyword. Hence, the ranking often contains redundant trees, with many overlapping triples. Moreover, these trees also have the same score, which results in a non-discriminating ranking. This is often the case with query keywords used in a dataset like 'actor' or 'director' for a database like LinkedMDB.

We re-implemented SUMM in order to have a new benchmark to perform keyword search system over bigger databases. We did not focus on efficiency-oriented aspects of the algorithm (e.g. the homomorphism among graphs used exploited to meet earlier the stopping condition), so our algorithm may be slightly slower in the on-line phase. However, the graphs building procedure is the same, hence the results in terms of effectiveness are the same as the original version of SUMM.

VI. TSA+VDP: DESCRIPTION AND FUNCTIONING

In this section, we describe the keyword search system we propose. This section expands and better describes the ideas already presented in [37] and [38].

A. OFF-LINE-PHASE - TSA

TSA (Topological Semantical Aggregator) is a virtual document-based approach that works off-line to build documents by aggregating RDF triples containing close concepts. TSA mimics the "clustering hypothesis" defined in IR, which dictates that triples characterized by similar concepts will cluster together. This concept is realized by the "subgraphs extraction module", which takes an RDF graph as input and creates subgraphs without considering the user query. It builds a subgraph around a single "topic" which characterizes the graph semantics. Within the scope of our example, a topic may be a film, an actor, or a director.

Definition 4: Let $\mathcal{G}$ be an RDF graph where $v \in \mathcal{G}$ is a node, $\delta^{-}(v)$ be the out-degree of v, $\delta^{+}(v)$ its in-degree, and $\lambda_{in}, \lambda_{out} \in \mathbb{N}$ be two threshold values. Then, all the nodes $v \in G \mid \delta^{-}(v) \geq \lambda_{out}$ are called *source nodes* and all the nodes $v \in G \mid \delta^{+}(v) \geq \lambda_{in}$ are called *terminal nodes*. The nodes that are neither source nor terminal nodes are called *accessory nodes*. We call *out-neighbourhood* of $v \in \mathcal{G}$, the set $\mathcal{N}^{-}(v)$ of the nodes at distance one from v.

Algorithm 1 TSA

```
Input : RDF dataset $\mathcal{G}$, source set $S$, terminal set $T$,
        list $\mathcal{L}$, integer radius $\tau$
Output: list $sbgr$ of representative subgraphs
$sbgr \leftarrow \emptyset$
$Q \leftarrow \emptyset$
foreach $s \in S$ do
    if $s.color = white$ then
        $Q.enqueue(s)$
        $s.\tau \leftarrow \tau$
        G ← new Graph()
        while $Q \neq \emptyset$ do
            $v \leftarrow Q.dequeue()$
            $v.color \leftarrow black$
            $radius \leftarrow v.\tau - 1$
            foreach $u \in \mathcal{N}^{-}(s)$ do
                if $u \notin S \wedge u \notin T$ then
                    // Accessory node
                    $G$.addTriple$((s, u))$
                if $u \in T$ then
                    // terminal node
                    $G$.addTriple$((s, u))$
                    foreach $w \in \mathcal{N}^{-}(u)$ do
                        if
                        $w.isLiteral() \vee (w \notin S \wedge w \notin U)$
                        then
                            $G$.addTriple$((u, w))$
                if $(p(s, u) \in \mathcal{L}) \wedge (radius >$
                $0) \wedge (u.color \neq black)$ then
                    if $u \in S \wedge u \notin T$ then
                        $u.\tau \leftarrow radius$
                        $Q.enqueue$(u)
                        $G$.addTriple$((v, u))$
        $sbgr$.add$(G)$
return $sbgr$
```

Algorithm 1 reports the pseudocode of TSA detailing the critical aspects of the algorithm in building the topic subgraphs. TSA takes an RDF graph $\mathcal{G}$, the sets of source and terminal nodes in $\mathcal{G}$ (i.e., S, T), a custom parameter $\tau \in \mathbb{N}^+$ defining the subgraphs radius, a list $\mathcal{L}$ containing all the predicates in $\mathcal{G}$ with a frequency higher than a given threshold (ignoring rare predicates in the database) as inputs. The output of TSA is a list of RDF subgraphs.

TSA iterates over the source nodes such that every node $s \in S$ becomes the starting point to create a subgraph using a BFS-like strategy. Given a source node s (extrapolated at random from S), TSA includes in the corresponding graph all the literals connected to s and all the neighboring terminal nodes along with their connected literals. Then the algorithm traverses all the edges to other source nodes. In particular, only the edges labeled with predicates that are contained in the list $\mathcal{L}$ are traversed. The exploration does not go beyond a certain radius τ user-defined, which controls the dimension of the subgraphs. Once the subgraph of s has been created (i.e., there are no more nodes satisfying the conditions), TSA colors s and all the other visited source nodes in black, meaning that they cannot be visited anymore, it then proceeds to build a new subgraph starting from the next non-black node in S. TSA stops when it has visited all the nodes in S. The resulting subgraphs are called *representative graphs* and the set containing them is called the *representative collection* of $\mathcal{G}$.

Example 3: Figure 8 shows an overview of TSA (left part) working on the running example. We set the following parameters: $\lambda_{in} = 1, \lambda_{out} = 4, \tau = 1$. With these parameters, the source nodes are: `F1`, `F3`, `A1`, `D1`.

The representative graph `G1` is then created starting from the node `F3`, the first one extracted from the set S, and then collecting its literals and terminal neighbours. Since there are no more source nodes at one hop from `F3`, TSA stops there and `F3` is coloured in black. `G2` is created starting from `D1` and including its literal node "Quentin Tarantino" and the terminal node `P2`. `F3` is a source node (already in `G1`), but since $\lambda_{in} = 1$, here it is considered as a terminal node and thus added to the graph. Then, within the radius of 1, TSA finds the source node `F1` and adds it to `G2` along with its literals and terminal nodes. `G3` is created with the same procedure. After it, there are no more source nodes left in the input graph with an out-degree greater or equal to $\lambda_{out} = 4$ and TSA ends.

The choice of the parameters λ_{in}, λ_{out} and τ affect the output representative collection. A big τ leads to big subgraphs because more source nodes will be inserted in the same subgraph. As a consequence, there will be fewer graphs in the representative collection. A big λ_{out} implies fewer source nodes, hence fewer subgraphs. A big λ_{in} leads to fewer terminal nodes to be added to the subgraphs and, therefore, to smaller graphs.

The on-line phase (right part of Figure 8) starts after the creation of the representative collection when the user query is taken into account. The on-line phase is composed of four steps. In step 1, we create an initial ranking of the virtual documents derived from the representative graphs by matching query keywords with the terms in the documents and determining the query-document similarity using the *BM25 function*. BM25 is one of the most effective probabilistic weighting schema used in IR. It is largely used for news retrieval and Web search.

Note that the first ranking may contain thousands of documents, especially if the input RDF graph is big, but we generally consider only the top part of the ranking, inside a certain threshold (usually equal to 10000). In this way, we use BM25 to retrieve the most promising (i.e., relevant) documents for a given query swiftly. This step is pivotal to lower the running time of the on-line phase because it allows the system to focus only on a small and promising set of candidate answers: the ones contained in the *first ranking*. From the selected documents, the system fetches the corresponding representative graphs and then processes them with a *merging function* (step 2), that seeks for overlapping triples. The merging function considers two subgraphs G_1 and G_2 as sets of triples. If the value $|G_1 \cap G_2|$ is greater than a certain threshold percentage h (e.g. h =30%), the two graphs are merged in a single one via set-based union.

Example 4: In Figure 8 we see that the cardinality of the intersection between `G1` and `G2` is of four triples. `G2` is the smaller graph, with 6 triples in total. So $\frac{2}{3}$ of its triples are overlapping. These graphs are merged by the system into `G1+G2`.

The merging algorithm proceeds as follows: it starts from the first graph of the ranking and considers the first l next graphs of the first ranking. The first graph is compared with the second. If the percentage of overlap is higher than h, then the algorithm merges the two graphs and removes them from the first ranking. With the new produced merged graph, the algorithm proceeds to the following graph in the ranking. This process is iterated for all the first l graphs of the ranking. Once this first iteration ends, the merged graphs are added to a new collection of *Merged Graphs*. The next iteration starts with the first graph, which was not merged to any other, until all the elements of the first ranking are considered or until the list of newly merged graphs reaches the dimension of 1000.

B. ON-LINE PHASE

Once the merged graphs collection is produced, we create the collection of virtual documents associated with these merged graphs. Then, we perform a second ranking using BM25 on these documents. We return the user the list of subgraphs in the ranking order produced by the BM25 function. We call this pipeline **TSA+BM25**, and it is composed by the TSA off-line phase and the on-line phase. The latter is composed of the merging function, followed by the ranking through BM25. This pipeline is quite fast in its on-line execution since it never requires to explore the graphs, but only relies on the algorithms of extraction of virtual documents and on the BM25 function.

The second pipeline is called **Virtual Document Pruning (VDP)**. It starts after the completion of the

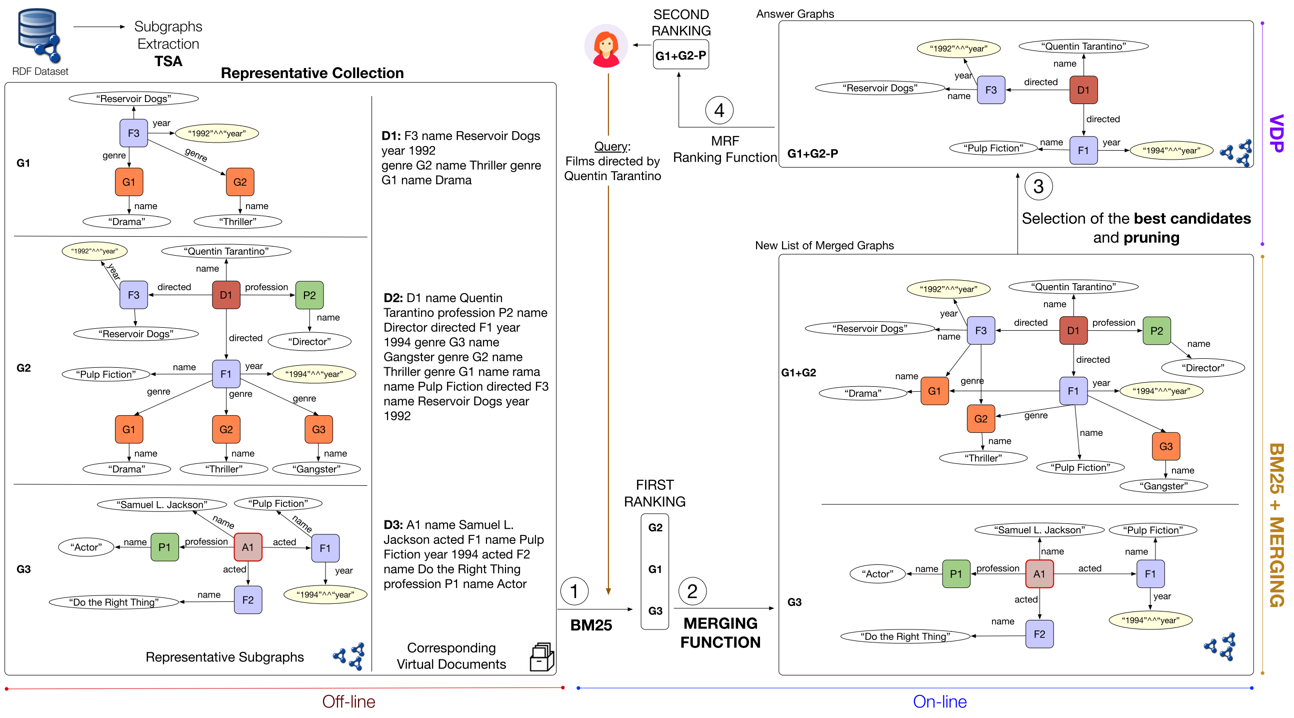
Subgraphs Extraction
TSA
RDF Dataset
Representative Collection
G1
G2
G3
D1: F3 name Reservoir Dogs year 1992 genre G2 name Thriller genre G1 name Drama
D2: D1 name Quentin Tarantino profession P2 name Director directed F1 year 1994 genre G3 name Gangster genre G2 name Thriller genre G1 name rama name Pulp Fiction directed F3 name Reservoir Dogs year 1992
D3: A1 name Samuel L. Jackson acted F1 name Pulp Fiction year 1994 acted F2 name Do the Right Thing profession P1 name Actor
Representative Subgraphs
Corresponding Virtual Documents
Query: Films directed by Quentin Tarantino
SECOND RANKING
G1+G2-P
MRF Ranking Function
Answer Graphs
VDP
Selection of the best candidates and pruning
New List of Merged Graphs
G1+G2
BM25 + MERGING
FIRST RANKING
BM25
MERGING FUNCTION
Off-line
On-line

FIGURE 8. Functioning of the TSA+VDP system based on the RDF graph shown in Figure 1.

TSA+BM25 pipeline. Given the merged graphs collection and the order provided by BM25 at the end of the TSA+BM25 pipeline, VDP considers the first n graphs of this ranking. These first n graphs are then considered together in a unique graph. The result of their set-based union is called *query graph* E. This graph is not necessarily connected. The parameter n allows a trade-off between effectiveness and efficiency in the ranking function. A big n leads to a bigger query graph E, which increases the probability of finding useful information at the expense of a longer running time.

VDP then starts a BFS from every subject node inside the query graph E. This BFS is limited to a radius τ and produces subgraphs. Of these subgraphs, only the ones containing all the keywords are kept, producing a new collection, called the *best candidates* collection. As these candidates are built from the query graph E and contain all the keywords, it is probable that they also contain a high number of relevant triples. Since the BFS exploration could have introduced many non-relevant triples, the next step is a pruning algorithm over these candidate graphs. The pruning proceeds inward, removing all the triples that do not contain at least one keyword, starting from the nodes with the highest distance from the source node.

Example 5: Figure 9 shows an example of how the pruning algorithm operates. It returns a smaller graph where triples not containing a keyword are removed. Keywords are highlighted in red. The numbers indicate the order in which the triples are removed from the graph, starting from the nodes with the highest radius from the source node where the BFS algorithm started (D1). The algorithm starts from a node and proceeds inward until the source node is reached or a triple containing at least one keyword is found. A triple, which is a component of a path from the root to a keyword, is not removed.

Step 4 of VDP produces the final ranking returned to the user. VDP employs the Markov Random Field module adapted from the one used in MRF-KS, described in Section V-B. Here, we consider unigrams and bigrams within the graph and the distance of the words from the root, but, differently from MRF-KS, we do not employ any prior, and we do not consider fields for the nodes. The score given to a subgraph g provided a query Q is given by the following Markov Random Field-based formula, derived from [14]:

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

where f_U and f_B are the functions for unigrams and unordered bigrams, respectively. For the unigrams, the function becomes:

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

where g^* is the virtual document obtained from the graph g and C is the collection of virtual documents obtained from the graphs of the representative collection. α_U is the Dirichlet's Smoothing factor, computed as $\frac{\mu}{\mu+|g^*|}$, where μ is the average length of the virtual documents from the representative collection.

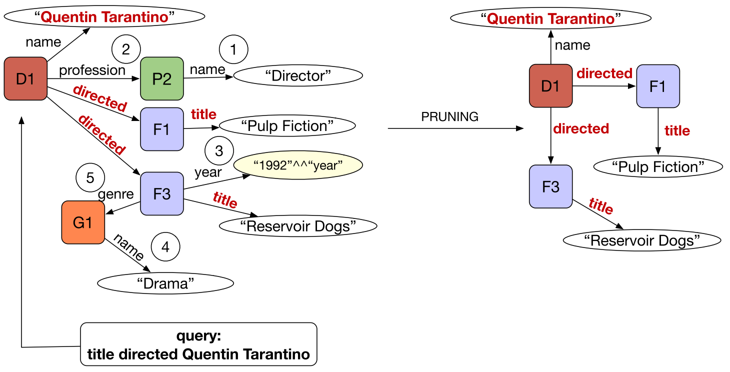

FIGURE 9. Example of the VDP pruning function. The numbers on the predicates indicate the order in which the algorithm removes the triples.

By using the Maximum Likelihood Estimation, the first and second probabilities become:

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

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

where *wtf* is a weighted term frequency that keeps into consideration not only the frequency of the term in the virtual document but also the position of the node where the word is contained in the graph. In turn, $tf(q_i, g^*)$ is the term frequency of the word q_i in the virtual document g^*. The *wtf* is based on the term frequency combined with a discount factor (a Gaussian kernel) considering the node position from the graph center. It is computed as follows:

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

where g is a graph, g^* is its virtual document, v is a node in g, v^* its associated document; s_g is a node in g which is considered the center of the graph (e.g. the starting node in the BFS exploration that created the graph); σ is a parameter that controls the spread of the kernel. Here $\sigma = 1$. The sum is over all the nodes v in g. The value $w_s(g^*, v)$ is called *relative static weight* of the node v in g^*, and is the minimum weight over all paths from s_g to v in g, that is:

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

The value $w_s(p_{s_g \to v})$ is called *static weight* of the path $p_{s_g \to v}$ from s_g to v. This weight is computed as follows:

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

Here $we_s(e)$ is the static weight of the edge e, and is usually set to 1, while $wn_s(x)$ is the static weight of the node x, determined as:

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

The bigram potential function is computed in a similar way.

Once all the nodes of the query graph E have been explored, the returned candidate answers are ranked using $score(g, Q)$ defined above and returned to the user.

VII. EVALUATION METHODOLOGY

The effectiveness and efficiency of keyword search systems need to be assessed. Effectiveness concerns the quality of the rankings returned to the user, whereas efficiency is about the off-line time required to index the dataset, the on-line time to execute a query and to return the final ranking to the user, the disk space occupied by the support data structures and the central memory required to answer a query.

The effectiveness of a keyword search system is usually evaluated using the Cranfield framework. The Cranfield framework adopts shared experimental collections expressed as a triple $\mathcal{C} = \{D, T, GT\}$ composed of a *dataset* D, a set of *topics* T, which simulate actual user information needs, and the *ground truths* GT, i.e. a kind of "correct" answers, where, for each topic, the relevant data are determined.

There are some critical differences between standard text retrieval and document-based approaches to keyword search over structured data. First of all, in standard text retrieval, the collection D of documents is fixed, and it is the same for all systems. In the document-based approach over structured data, the documents are created dynamically with different strategies. Hence, we cannot assess the relevance of a document to a query *in advance* as we do for standard text retrieval. We can, however, assess the relevance of a given answer only when the system returns it. An immediate consequence of this implies that, while within the Cranfield framework the ground truth GT is usually manually built by human assessors that establish whether a document is relevant for a given topic, in keyword search over structured data this is not possible since the documents to be judged are created on the fly and vary with the query and the search system employed.

We, therefore, need to find a way to automatically build a reliable GT that enables an assessment of whether a dynamically produced document is relevant or not after its generation.

As a consequence, a keyword query can be rewritten as a SPARQL query by a domain and RDF expert. A SPARQL query returns the "exact" answer to a topic, and this answer can be identified as a "ground truth" for the given topic. So a SPARQL query Q_{t_k} associated to a given topic $t_k \in T$, returns the "perfect answer graph" – i.e. the *ground truth graph* $\mathcal{G}_{t_k}$ – to the topic. We deem as "relevant" to the topic t_k all the triples in the answer graph $\mathcal{G}_{t_k}$. Conversely, all the triples outside $\mathcal{G}_{t_k}$ are deemed as "not-relevant".

Definition 5: Let Q_k be a keyword query derived from the topic $t_k \in T$ and submitted to a keyword search system and let $\mathcal{G}_{t_k}$ be the ground truth graph of t_k. We define the ranking returned by the system as $R_k = [ap_1, ap_2, \dots, ap_n]$, where $ap_i = (\mathcal{G}_i, sim_i) \in R_k, i \in [1, n]$ is called *answer pair* s.t. $\mathcal{G}_i$ is the RDF answer graph at rank i and $sim_i \in \mathbb{R}$ is a degree indicating the similarity of $\mathcal{G}_i$ to $\mathcal{G}_{t_k}$.

For every couple of answer pairs $< ap_i, ap_j >$ in R_k s.t. $i < j$, it holds that $sim_i \geq sim_j$.

Hence, a sound keyword search system should return a ranking that is: (i) *top-heavy*: the answer graphs ranked at top ranks contain more relevant triples than noisy triples; and, (ii) *essential*: the answer graphs ranked at the bottom should not contain any triples (relevant or not relevant) already returned by the graphs ranked at the top (i.e., redundant triples spoil the ranking quality).

First of all, it is necessary to establish when a graph is relevant. One possibility is to consider the number of relevant and non-relevant triples inside it.

Definition 6: Given a topic $t_k \in T$, a ranking R_k and the ground truth graph $\mathcal{G}_{t_k}$, we define the *Signal-to-Noise Ratio* (SNR) of $\mathcal{G}_i \in R_k$ as

$$SNR(\mathcal{G}_i) = \frac{|(\mathcal{G}_i \cap \mathcal{G}_{t_k}) \setminus S|}{|\mathcal{G}_i|}$$

where S is the union set of all the relevant triples in $\mathcal{G}_j \in R_k, \forall j \in [1, i[$.

The SNR of a graph $\mathcal{G}_i$ rewards precise and essential graphs as it decreases whenever the graph contains non-relevant triples. The SNR proxies a sort of precision since the numerator is equal to the number of relevant triples found for the first time in the ranking, while the denominator is the total number of triples of the graph. We can say that a graph is relevant to the information need of a user when its *SNR* is bigger or equal to a threshold value of $\lambda \in [0, 1]$. We call λ the *relevance parameter*, which describes the quality of a graph to be relevant. In this evaluation framework, the relevance is not an absolute, fixed concept, but it varies based on the task at hand. In some cases, relevance can be strictly determined (high λ); in others, it can be more slack (low λ).

Once a graph is deemed relevant or non-relevant, we can consider different metrics to evaluate the quality of a ranking. This paper considers three evaluation metrics, as defined below.

Definition 7: The *recall* of a ranking R_k is defined as:

$$recall(R_k) = \frac{|\bigcup_{\mathcal{G}_i \in R_k \mid SNR(\mathcal{G}_i) \geq \lambda} (\mathcal{G}_i \cap \mathcal{G}_{t_k})|}{|\mathcal{G}_{t_k}|}$$

The recall is obtained by the ratio between the total number of relevant triples found in the relevant graphs in a ranking and the cardinality of the GT. It describes the system's ability to extrapolate the relevant triples from the database to be searched. It does not penalize relevant lower-ranked triples and does not provide any information about the quantity of noise that is present in the relevant answer graphs. In the following, we always use a recall computed over the first 1000 answer graphs of the rankings produced by the systems. In some cases when the rankings have less than 1000 graphs, the recall will be computed over all the returned answer graphs.

Definition 8: The *precision* of a ranking R_k is defined as:

$$precision(R_k) = \frac{|\bigcup_{\mathcal{G}_i \in R_k \mid SNR(\mathcal{G}_i) \geq \lambda} (\mathcal{G}_i \cap \mathcal{G}_{t_k})|}{|\bigcup_{\mathcal{G}_i \in R_k} \mathcal{G}_i|}$$

Precision is the ratio between the total number of triples found in the relevant graphs of a ranking and the total number of distinct triples in the ranking. Precision also takes into account the noise in the ranking. The higher the number of non-relevant retrieved triples, the bigger the denominator, and thus the smaller the precision. A good ranking yields graphs that are well-tailored around relevant triples that are with as few non-relevant triples as possible, and with all the relevant triples in the top positions.

Since the denominator is the number of all of the distinct triples contained in the ranking graphs and the numerator can reach at max the cardinality of the GT, the values assumed by the precision can quickly tend to 0 when hundreds of elements compose the ranking.

Precision is thus estimated at different levels of cut-off, that is, only considering the top c elements of the ranking.

Definition 9: We define the *prec@c* the precision computed at level of cut-off c:

$$prec@c(R_k) = \frac{|\bigcup_{\mathcal{G}_i \in R_k \mid SNR(\mathcal{G}_i) \geq \lambda \,\wedge\, i \in [1,c])} (\mathcal{G}_i \cap \mathcal{G}_{t_k})|}{|\bigcup_{\mathcal{G}_i \in R_k \mid i \in [1,c]} \mathcal{G}_i|}$$

In particular, we study prec@1 and prec@5. While prec@1 only considers the first graph of the ranking, prec@5 considers only the top 5. These two precisions describe the system's ability to extrapolate relevant triples without too much noise in the top positions. A system that inserts too much noise, or that does not rank graphs containing relevant triples in the top positions, will determine low values of precision.

Both of these two measures are not top-heavy. They, therefore, do not weight the position of the relevant triples within the ranking.

Definition 10: We define the *Graph Relevance Weight* (GRW) of $\mathcal{G}_i \in R_k$ as

$$GRW(\mathcal{G}_i) = \frac{|(\mathcal{G}_i \cap \mathcal{G}_{t_k}) \setminus S|}{|\mathcal{G}_{t_k}|}$$

where S is the union set of all the relevant triples in $\mathcal{G}_j \in R_k, \forall j \in [1, i[$.

The GRW is a weighting function that rewards the graphs that contain many relevant triples seen for the first time in the ranking – i.e., GRW rewards the essentialness of a graph.

We use GRW as the central component of a new measure, called *triple-based Discounted Cumulative Gain* (tb-DCG). It awards the top-heaviness and the essentialness of a ranking and is a variation of the DCG measure [39] widely-used in text retrieval (especially for Web search tasks).

Definition 11: Let $t_k \in T$ be a topic, R_k a ranking with length $n \in \mathbb{N}^+$, $\lambda \in [0, 0.1, 0.2, \ldots, 1]$ be a threshold value and $b \in \mathbb{N}^+$ a logbase value. We define the *Relevance Gain* (RG) of $\mathcal{G}_i \in R_k$ as

$$RG_b(\mathcal{G}_i) = \begin{cases} GRW(\mathcal{G}_i) & \text{if } i \leq b \ \wedge \ SNR(\mathcal{G}_i) > \lambda, \\ \dfrac{GRW(\mathcal{G}_i)}{\log_b i} & \text{if } i > b \ \wedge \ SNR(\mathcal{G}_i) > \lambda, \\ 0 & \textit{if } SNR(\mathcal{G}_i) \leq \lambda. \end{cases}$$

The *triple-based Discounted Cumulative Gain* (tb-DCG) of the ranking R_k is defined as

$$tb\text{-}DCG_b(R_k) = \sum_{i=1}^{n} RG_b(\mathcal{G}_i)$$

We can see that the graphs returned to top positions (i.e. $i \leq b$) provide a full informative gain to the user (i.e. $GRW(\mathcal{G}_i)$). In turn, the gain provided by the graphs down in the ranking is discounted by a factor growing with the position rank (i.e. $\log_b i$). The logbase b models the probability that a user will consult (or click) an answer graph returned at a given position i; the higher the i rank of a graph, the lower the probability a user will consult the graph and thus the lower the overall usefulness of the search system. tb-DCG is within the [0, 1] interval, where 0 indicates a system returning no relevant triple to the user and 1 indicates a system returning the ground truth graph within the first b positions of the ranking.

Example 6: With reference to query Q of Example 1. Its ground truth is the graph $\mathcal{G}$ in Figure 2. As we can see, $|\mathcal{G}| = 5$.

Now consider Figure 10. We see the ground truth graph GT and three graphs ranked by a system in this order: GA1, GA2, GA3. We consider $\lambda = 0.7$ and $b = 2$. GA1 presents 3 relevant triples, so its SNR is 0.75. The graph is considered relevant. Its RG is $\frac{3}{5} = 0.6$. Graph GA2 yields one relevant triple (A-E) and a second triple (A-C), in blue, which is already contained in GA1. Thus it is not counted as relevant. GA2 presents a SNR of 0.5, below the threshold λ. Its RG is therefore 0. The last graph, GA3, contains two relevant triples that are not contained in GA1. Thus its SNR is 1. Its RG is given from the ratio $\frac{2/5}{log_2 3} \approx 0.25$. The division by logarithm is the discount, due to the third position in the ranking. This ranking has an overall $tb\text{-}dcg_2$ of $0.6 + 0 + 0.25 = 0.85$.

If GA3 were put in the second position of the ranking, its RG would have been $\frac{2}{5}$, and so the $tb\text{-}DCG$ of the ranking would have been 1. This means that in a best-case scenario, that is, when all the relevant triples are covered by the first b graphs of the ranking, the $tb\text{-}DCG$ sums to 1, meaning that the ranking is *optimal*: it provides to users all the required information inside the first b graphs. The value of the SNR describes the quality of the required answers, i.e., how much noise the user can accept inside a graph. The value of b represents the user's patience: the bigger b, the longer the user is willing to look into the list for relevant triples. When a relevant graph is placed in rank beyond the threshold b, its relevance suffers the discount.

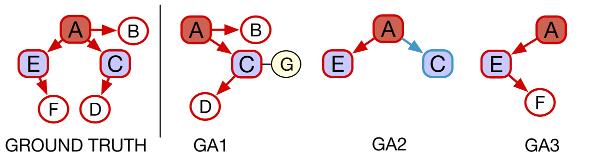

FIGURE 10. Example of a ranking.

VIII. EXPERIMENTAL SETUP

As discussed above, [7] defines a shared test collection to evaluate the effectiveness of search systems over relational data based on three datasets, but there are no shared test collections for RDF. Hence, we have created test collections based on real and synthetic RDF datasets.

A. REAL DATABASES

As real databases, we employed LinkedMDB [40], IMDB[8] (also adopted in [7]) and a subset of DBPedia as defined in [41] for entity search.

LinkedMDB is a native RDF database of approximately 7M triples. Whereas, IMDB is a relational database that we converted into an RDF dataset of about 116M triples.[9]

For LinkedMDB and IMDb, we designed 100 topics where half were used for training and a half for testing. A topic is composed of three fields: the `title`, the `dsc` (description) and the `SPARQL query`. The `title` is a short string of text which summarizes the information need with few keywords, used as keyword query. The `dsc` is a detailed description of the information need used to characterize it better. The `SPARQL query` contains the SPARQL that returns the "correct" answer graph to answer the information need. All queries return an RDF graph built by using the `construct` SPARQL query type.

Regarding DBPedia, we have drawn a subgraph of the dataset following the procedure described by [41].[10] Specifically, the DBPedia subgraph is composed of the DBPedia Ontology, the Ontology Infobox Types, the Titles subset, the Short Abstract subset, and the Raw Infobox Properties subset. The resulting RDF graph has about 70M triples. As for the queries, we considered 50 topics from the classes `QALD2_te` and `QALD2_tr` used by [41].[11] These two

[8]https://datasets.imdbws.com/

[9]We selected a subset of the files provided in the IMDB webpage, namely `name.basics`, `title.basics`, `title.crews` and `title.principals`.

[10]https://wiki.DBPedia.org/data-set-37

[11]https://iai-group.github.io/DBPedia-Entity/

TABLE 1. Name of the databases used with their dimension in (millions) of triples and the number of queries used in our experiments.

Database name	type	number of triples	number of queries
LinkedMDB 1M	real	1M	50
IMDB 1M	real	1M	50
LUBM 1M	synthetic	1M	14
BSBM 1M	synthetic	1M	13
LinkedMDB	real	7M	50
LUBM 10M	synthetic	10M	14
BSBM 10M	synthetic	10M	13
IMDB	real	116M	50
DBPedia	real	70M	50

classes of queries are in natural language form and are used in [41] for an entity search task. Given that these topics and their associated ground truth work for entity search tasks, we cannot use them directly for a keyword search. We manually mapped them into `construct` SPARQL queries and derived the corresponding keyword queries to use them in our evaluation framework.

B. SYNTHETIC DATABASES

Regarding the synthetic databases, we used the Lehigh University Benchmark (LUBM)[12] [42] and the Berlin SPARQL Benchmark (BSBM)[13] [43]. LUBM is a database about universities, professors and students developed by the Lehigh University to facilitate the evaluation of Semantic Web Repositories in a standard and systematic way. The benchmark currently provides 14 SPARQL test queries.[14] We took these queries, converted them in `construct` versions, and produced their equivalent keyword query, extrapolating distinct keywords from the words contained in the SPARQL query.

BSBM is a database built on an e-commerce use case, where different vendors with posted reviews offer a set of products. BSBM defines three use cases with different queries that focus on different aspects of the SPARQL query language. We used the Explore use case,[15] with 13 different SPARQL queries. We converted these queries into `construct` SPARQL queries as we did for LUBM.

The APIs of these databases served to create versions of datasets of different dimensions. We created two versions for each of the two synthetic datasets of 1 and 10 millions of triples each.

Table 1 reports information about the databases used with their number of triples and the number of queries we used as a test set. As we can see, the datasets can be divided into three categories based on the magnitude of their dimensions. We refer to these categories as 1M, 10M, and 100M.

[12]http://swat.cse.lehigh.edu/projects/lubm/

[13]http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/

[14]http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt

[15]http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/ExploreUseCase/index.html

C. EXPERIMENTAL SETTINGS

All our experiments are run on a Linux/4.8.13-100.fc23.x86_64 amd64 Intel(R) Xeon(R) CPU X5680 with 3.33GHz Family 6 Model 44 Stepping 2, GenuineIntel and 24 CPU. All the code is written in Java. We use Blazegraph[16] to manage the RDF graphs, PostgreSQL 10.5[17] to manage additional data extracted from graphs and Terrier v4.2[18] [44] as indexing and retrieval methods.

Each RDF dataset is represented with a relational table in PostgreSQL, made of three fields: subject, predicate, object. One index is applied to the field subject for a quick exploration of the graph. We preferred PostgreSQL because it showed better performances than Blazegraph when exploring the graphs. The Blazegraph library was used to perform the SPARQL queries that produced the GT, to build the smaller answer graphs in central memory, and to read/write them in the secondary memory. Finally, Terrier is used for its state-of-the-art implementation of BM25 and indexing methods in different phases of our pipelines.

In general, we set a maximum off-line execution time of 48 hours and a max on-line execution time of 1000 seconds, unless otherwise specified. If the system has not concluded within the time limits, we stop its execution and mark it with a timeout exception.

For reproducibility purposes, the source code of the search systems, the scripts to obtain the test datasets (both full and reduced) and the code to run the experiments along with the topics and the keyword queries, are available at the URL: `https://bitbucket.org/account/user/keywordsearchrdfproject/projects/TSAC`.

IX. EXPERIMENTAL EVALUATION

This Section is composed of three parts based on the database size considered: small (1M), medium (10M), and big (100M) datasets. We examine the performances of the five keyword search systems described in Section V. We measure their effectiveness with the measures of tb-DCG, recall, prec@1, and prec@5 computed on average over all the queries. To evaluate the efficiency, we measure the off-line and on-line execution time and the central memory employed by the systems.

A. SMALL DATASETS (1M TRIPLES)

Table 2 reports the values of tb-DCG, recall, prec@1, prec@5, on-line time, and memory for the five systems on the four databases of 1 M triples.

Let us start with the real databases LinkedMDB 1M and IMDB 1M. All the evaluation metrics reported in this Section for these databases were computed using $\lambda = 0.1$. Regarding tb-DCG, we can see how the top-performing systems are TSA+VDP and MRF-KS, whereas TSA+BM25, SUMM, and SLM present marked lower performances.

[16]https://www.blazegraph.com/

[17]https://www.postgresql.org/

[18]http://terrier.org/docs/v4.2/

TABLE 2. **Performances of the algorithms on the 1M databases.** † **indicates the top performing systems with** $\alpha < 0.01$ **accordingly to the Tukey's HSD test. The best system is in bold.**

Dataset	Systems	tb-DCG	recall	prec@1	prec@5	time (sec)	memory (MB)
LinkedMDB 1M	TSA+BM25	0.201±0.02	**0.733±0.02**†	0.049±0.01	0.032±0.01	**39.64±01.90**†	13.19±0.41
	TSA+VDP	**0.490±0.04**†	0.592±0.040†	0.177±0.02†	0.186±0.02†	318.78±21.60	21.06±1.18
	SLM	0.011±0.00	0.076±0.01	0.040±0.01	0.012±0.00	39.90±08.69†	**0.82±0.22**
	MRF-KS	0.400±0.03†	0.462±0.03	0.145±0.01	0.131±0.01	285.22±30.10	0.99±0.09
	SUMM	0.106±0.01	0.268±0.02	**0.272±0.02**†	**0.226±0.02**†	429.52±37.17	20.54±1.20
IMDB 1M	TSA+BM25	0.276±0.02	0.423±0.03	0.337±0.04	0.192±0.02	30.28±00.25†	7.31±0.07
	TSA+VDP	**0.583±0.04**†	**0.623±0.04**†	0.315±0.03	0.265±0.03	196.90±20.69	12.27±0.82
	SLM	0.011±0.00	0.057±0.00	0.04±0.01	0.028±0.00	**15.92±06.57**†	0.40±0.16
	MRF-KS	0.312±0.02	0.382±0.02	0.451±0.03†	0.275±0.03	399.68±37.34	**0.22±0.02**
	SUMM	0.126±0.01	0.254±0.02	**0.504±0.03**†	**0.484±0.03**†	510.26±27.02	17.27±0.78
LUBM 1M	TSA+BM25	**0.284±0.06**†	0.344±0.07	0.120±0.04	0.131±0.04	**35.28±20.36**†	18.59±9.67
	TSA+VDP	0.243±0.05†	0.091±0.04	**0.224±0.06**	**0.306±0.05**†	406.64±90.74	19.44±9.69
	SLM	0.048±0.00†	**0.578±0.07**†	0.142±0.06	0.201±0.06†	61.14±28.84†	**1.71±0.94**
	MRF-KS	0.090±0.03†	0.380±0.07†	0.000±00	0.000±0.00	304.86±61.54†	2.00±0.40
	SUMM	0.024±0.01	0.231±0.06†	0.135±0.05	0.122±0.05	526.14±65.67	11.11±1.41
BSBM 1M	TSA+BM25	**0.441±0.07**†	**0.837±0.06**†	0.066±0.02	0.084±0.02	**57.61±04.77**†	148.71±11.33
	TSA+VDP	0.231±0.06†	0.439±0.08	0.131±0.04	**0.132±0.04**	293.15±17.50†	153.84±11.29
	SLM	0.025±0.01	0.037±0.01	0.076±0.05	0.076±0.05	417.69±48.84	23.07±03.32
	MRF-KS	-	-	-	-	-	-
	SUMM	0.109±0.04	0.120±0.05	**0.190±0.05**	0.122±0.03	422.54±84.78	**20.20±04.50**

In particular, the difference between TSA+VDP and TSA+BM25 shows the critical contribution of the pruning algorithm and ranking function of VDP. We can see that for both LinkedMDB 1M and IMDB 1M, TSA+VDP shows a remarkable improvement in the performance compared to the ones obtained by TSA+BM25. This improvement results from the VDP strategy, which is particularly useful in real datasets. MRF-KS works well with LinkedMDB 1M without a significant statistical difference compared to TSA+VDP. Nevertheless, it performs substantially worse for IMDB 1M, and similarly to TSA+BM25. Moreover, while the two TSA-based systems improved their performance, passing from LinkedMDB 1M to IMDB 1M, MRF-KS shows a lower tb-DCG. In LinkedMDB 1M, the systems TSA+VDP and MRF-KS belong to the top group, while for IMDB 1M, TSA+VDP is the only system in the top group. In IMDB 1M, MRF-KS decreased its value of tb-DCG and is in the same class as TSA+BM25. Not all queries could finish in time for this system in this database. This was due to a more connected structure, which on average, requires more time to complete the execution of the Dijkstra algorithm. The same experiments were performed without a time limit, and in this case, MRF-KS scores a tb-DCG equal to 0.434, yielding a noteworthy improvement in performance, but still way below that of TSA+VDP.

TSA+BM25, SLM, and SUMM do not obtain the same results as our two top-performing systems. TSA+BM25 does not perform any pruning on the ranking obtained by the BM25 function, resulting in noisier and non-relevant graphs. Moreover, the ranking function works only on virtual documents and does not exploit the graph nature of the answers. This has repercussions on the quality of the ranking.

Regarding SLM, its graph building strategy relies too much on matching words from the keyword queries with words in the triples. During the process, a new triple is added to an answer graph only if it contains a set of keywords that differ from the ones of the other triples in the graph. In this way, the answer graphs are often small, with overlapping triples, and they do not contain all of the keyword queries. This leads to a poor ranking.

Regarding SUMM, the results are determined by its answer-building and ranking strategy. The tree answer graphs produced by SUMM can be potentially very similar to each other, with many overlapping paths from the root to the leaves. This leads to a significant overlap within the system answers, that is, redundancy among the answers. Moreover, since the weighting function of SUMM relies only on the distances of the paths, often, this value is non-discriminant.

In considering the recall for the two real datasets, TSA+BM25 has the highest values among all of the systems, with a recall above 0.7. TSA+VDP comes close in the second position, but statistically speaking is in the top group. MRF-KS, in this case, is in the third position. This shows how TSA+BM25 extracts many relevant triples from LinkedMDB 1M on average. If we compare this result to its tb-DCG, it appears that TSA+BM25 extracts many relevant triples, but it is less effective in ordering the graphs containing them. TSA+VDP, in comparison, extracts less relevant triples, but it returns a better ranking.

Moving to IMDB 1M, TSA+VDP now presents the highest recall and is the only one in the top-performing group. TSA+BM25 and MRF-KS present lower results, which are now similar one to the other. This is new evidence that highlights how the nature of the database can genuinely change the performance of a system. In this case, the pruning ability of TSA+VDP removes irrelevant triples from the returned graphs, ensuring a higher SNR. This, in turn, implies that the number or answer graphs considered relevant is greater and, therefore, the overall number of relevant triples found is higher.

Focusing now on prec@1, the best performing system for both the real databases is SUMM. The high precision of this system can be directly associated with its answer-building strategy. SUMM builds answer trees with keyword nodes as leaves. In this way, the number of non-relevant triples is usually minimal, while the number of relevant ones is maximal. This strategy causes much redundancy in the answers of the ranking, as explained in Section V-C. Consequently, it is not the best whenever we are searching for high recall values. On the other hand, it is the most suited when we are searching for high precision in our answers. When considering the values of tb-DCG, which also takes into consideration the redundancy of the answers and their ranking, SUMM has the lowest values after SLM. This means that a user can find precise answers at the top of the ranking, but will not find all the relevant triples required to satisfy her information need since they are scattered down in the ranking or are lost in noisy answer graphs.

The answer-building strategy of MRF-KS is still based on the idea of building answer trees. It is, therefore, unsurprising to see high values of prec@1 for this system on both real databases. In particular, on LinkedMDB 1M, SUMM is in the top-performing group together with TSA+VDP, whose pruning strategy guarantees excellent precision. Whereas, for IMDB 1M, SUMM is in the top group together with MRF-KS. Generally speaking, the TSA-based algorithms work with lower precisions because they create bigger answer graphs than those of SUMM. This results in lower prec@1, but in higher recall and, thanks to the ranking function, also in higher tb-DCG.

Performances are similar if we consider prec@5. SUMM is still the top-performing algorithm on both the real databases. The performances of MRF-KS now decreases on IMDB, making SUMM the only system in the top-performing group.

In considering the time required on average to answer a query, we can see that SLM and TSA-BM25 are the fastest algorithms on both the real databases. They always belong to the top-performing group. The high efficiency of SLM is obtained thanks to its strategy in generating answer graphs, which limits their number. TSA+BM25 is fast because it relies on the state-of-the-art implementation of BM25 and does not employ time-consuming on-line algorithms on graphs. TSA+VDP requires more time than TSA+BM25 since it operates a re-ranking of the first ranking obtained by the previous system. The system compensates the higher execution time (which is still limited thanks to the parameter n, which controls the space of possible answers) with higher tb-DCG in its results. MRF-KS creates the answer graphs on the fly using the Dijkstra algorithm, which usually requires time. Note that while in [15] the parameter k was set to 5, here we set it to 1000. This explains the higher on-line execution time. SUMM is always the slowest algorithm since its stopping condition is not always met: many queries incur in the 1000s time limit.

Now let us consider the synthetic datasets. Here we use $\lambda = 0$ as evaluation parameter. In this way, we do not negatively weigh the noise in the answers of the systems, and thus, every answer containing at least one relevant triple is considered relevant. The synthetic databases are harder for all of the systems, and the values of effectiveness quickly flatten to 0 when λ increases. The nature of the synthetic datasets explains this. These graphs consist of a limited lexicon in their nodes and edges. LUBM builds nodes and edges using incremental numbering. We, therefore, have nodes containing words like 'University$x$' and 'Professory' where x and y are numbers incremented automatically. BSBM uses a limited database of words to build synthetic names of products and their reviews. Since all of the studied systems rely on the query-subgraph word matching, a limited lexicon has a significant impact on the overall quality of the output. Consider, for example, the query 'Student attending University0' for LUBM 1M, which asks for all the students attending the university called University0. The keyword 'student' is often used in LUBM and is found in many virtual documents and answer graphs produced by the systems. Even the non-relevant graphs contain the keyword 'Student,' sometimes with high frequency. This limited lexicon leads to answer graphs of lower quality since they contain many non-relevant triples. For both these reasons, the quality of the rankings decreases rapidly when λ increases, since fewer and fewer graphs are considered relevant. Thus, we use $\lambda = 0$ to highlight differences among systems.

In LUBM 1M, TSA+BM25 and TSA+VDP are the two top-performing systems in terms of tb-DCG. The other systems are now far below with lower values.

TSA+BM25 performs even better on average on BSBM. It is in the top class with TSA+VDP. In this case, we do not have data for MRF-KS since the system did not finish the off-line phase in time.

Let us consider recall and precision. SLM is the top-performing system for recall, in the first-class together with MRF-KS. TSA+BM25 is in the third position, behind MRF-KS. This indicates how the SLM strategy, in this database, retrieves more relevant triples in the ranking. However, as seen with the values of tb-DCG, the system does not rank them appropriately to meet the user needs.

On the other hand, TSA+VDP has the lowest recall among all the systems. Despite its low performance, the ranking function yields a good ranking, generating a good tb-DCG, second only to TSA+BM25.

For BSBM 1M, the situation is quite different. TSA+BM25 obtains a high recall close to 0.85, and it is the only one in the top-performing group. TSA+VDP is the other top-performing system, with a value of 0.439. The execution of all other systems produced much lower values of recall.

Concerning prec@1, for LUBM 1M, all the systems, except for MRF-KS, which gets a 0, perform similarly.

TABLE 3. Report of the off-line times required by different algorithms on the 1M datasets.

Database	algorithm	off-line time (min)
LinkedMDB 1M	TSA	16
	SLM	1
	MRF-KS	1,360
	SUMM	42
IMDB 1M	TSA	9
	SLM	1
	MRF-KS	3,117
	SUMM	54
LUBM 1M	TSA	94
	SLM	1
	MRF-KS	1,440
	SUMM	2,139
BSBM 1M	TSA	21
	SLM	2
	MRF-KS	-
	SUMM	107

TSA+VDP is the top system, followed by SLM and TSA+BM25. Considering prec@5 for the same database, we see an increase in values for TSA+VDP and SLM. These systems retrieve more relevant triples in the top 5 positions. The others, TSA+25, SUMM, and MRF-KS, show no significant changes when passing from prec@1 to prec@5. Worth noting that MRF-KS is a strategy that does not retrieve relevant triples in the top 5 positions on this database. This is also reflected in the low tb-DCG obtained by the system.

In considering now BSBM 1M, the situation changes slightly. SUMM now becomes the top system for prec@1, followed by TSA+VDP, SLM, and TSA+BM25. Moving to prec@5, the value of precision of SUMM decreases, slightly below TSA+VDP: the introduction of too many triples reduces the precision.

Concerning time, TSA+BM25 is still the fastest algorithm on both of the synthetic databases: in the top-performing group together with SLM in LUBM and with TSA+VDP for BSBM.

The last information presented in Table 2 is the average memory required by the systems. As we can see, all of them maintain the same quantity of memory throughout the first three databases, while, interestingly, BSBM 1M requires much more memory for TSA+BM25 and TSA+VDP. This is due to the higher number of potential answer documents found by BM25. Once again, this may be an effect of the high repetition of certain words in the BSBM database.

Table 3 reports the off-line time required by the different algorithms on the various 1M datasets. As we can see, the fastest algorithm is always SLM, requiring only 1 minute. SLM, in its off-line phase, only converts all the triples of the graph into virtual documents and index them. This can be done efficiently with the state-of-the-art implementations in Terrier.

TSA is always the second fastest algorithm. Exploring the graph by using the BFS based algorithm described in Section VI proved to be very fast. It completed the creation and indexing of the representative collection with its virtual documents in tens of minutes. Interestingly, LUBM 1M requires more time than the other three databases, probably due to its more connected structure.

SUMM is the third algorithm in terms of off-line time. Even if its strategy is still BFS-based, it requires keeping track of information such as the portal nodes and the partitions connected by them, with the help of a relational database. We did not implement the indexing algorithm for the subgraphs described in [15] based on homomorphism among graphs. This version of the algorithm is, therefore, probably faster off-line and slower on-line.

MRF-KS appears to be the slowest system in the off-line phase since the numerous Dijkstra explorations required time to build virtual documents and obtain information on distances among nodes. This can be assumed as the main reason why MRF-KS cannot be used over bigger databases. The creation of the virtual documents would require many days, if not months, over real word databases.

B. MEDIUM DATASETS (10M TRIPLES)

In this section, we report the results obtained with the experiments on LinkedMDB 7M, LUBM 10M, and BSBM 10M.

Table 4 reports the results from the three databases. Only three systems scale to this dimension: TSA+BM25, TSA+VDP, and SUMM. All three are based on the virtual-document approach. We use the parameter $\lambda = 0$ to estimate the relevance of the ranking and show the systems' different behavior over these datasets. For the synthetic databases, the justification is the same as the one provided in the previous section. In the real database LinkedMDB, which is is not a subgraph like LinkedMDB 1M, the level of connectivity among the nodes is higher. On average, the answer graphs extracted by TSA and SUMM are bigger and contain more noisy triples.

Starting from LinkedMDB 7M and tb-DCG, we see that TSA+VDP is the top-performing system and the only one in the top-performing statistical group. The values of TSA+BM25 and SUMM are far below. At this dimension, the pruning and re-ranking heuristics of TSA+VDP work very well, with high results of tb-DCG.

SUMM, on the contrary, is quite ineffective, scoring a tb-DCG of 0.049. A bigger database implies greater redundancy in the set of potential answers trees produced by the algorithm and in overlap among the paths. This effect was already discussed for the 1M databases, but here it is exacerbated by the bigger dimension.

Moving to the values of recall, TSA+BM25 presents a value of 0.916. The system is the only one in the top-performing group. This means that most of the relevant triples are retrieved and that the dataset dimension does not hinder the information extraction process. TSA+VDP has a lower value of recall for the very same reason as discussed in the previous session: in fact, it only works on the top $n = 100$ graphs of the ranking of TSA+BM25. For SUMM, the recall is lower due to the high overlap of triples in the ranking.

If we consider now the values of prec@1, TSA+VDP shows the highest value and is in the same group with SUMM.

TABLE 4. Performances of the algorithms on the 10M databases. Relevance decided with $\lambda = 0$. † indicates the systems in the top performing group with $\alpha < 0.01$. The best system is in bold.

Dataset	Systems	tb-DCG	recall	prec@1	prec@5	time (sec)	memory (MB)
LinkedMDB 7M	TSA+BM25	0.171±0.01	**0.916±0.02**†	0.06±0.00	0.003±0.00	**87.52±12.86**†	**23.24±00.55**
	TSA+VDP	**0.429±0.04**†	0.458±0.08	**0.037±0.00**†	**0.036±0.00**†	425.30±42.26	45.88±03.22
	SUMM	0.049±0.01	0.119±0.01	0.018±0.00†	0.017±0.00†	741.16±25.78	37.03±00.76
LUBM 10M	TSA+BM25	**0.281±0.07**†	**0.505±0.07**†	0.082±0.04†	0.111±0.05†	**29.57±8.27**†	**26.42±14.72**
	TSA+VDP	0.234±0.06†	0.384±0.03†	0.145±0.03†	**0.226±0.05**†	37.42±10.14†	26.71±14.72
	SUMM	0.053±0.00	0.123±0.04	**0.159±0.05**†	0.116±0.04†	794.29±79.99	10.93±01.60
BSBM 10M	TSA+BM25	**0.139±0.05**†	**0.227±0.07**†	0.039±0.01†	0.010±0.00†	**53.69±4.83**†	**146.01±13.89**
	TSA+VDP	0.074±0.03†	0.047±0.03†	**0.071±0.03**†	**0.071±0.03**†	70.38±7.70†	147.15±13.94
	SUMM	0.0001±0.00	0.018±0.00	0.013±0.00†	0.011±0.00†	612.77±79.99	17.60±04.30

This performance may be reconducted to the pruning strategy of the algorithm. The values are, however, quite low because of the high number of noisy triples in these first answers. In the case of prec@5, the situation does not change much, with TSA+VDP still as the top-performing system within the top group with SUMM.

Focusing now on the synthetic databases, in LUBM 10M, TSA+BM25 is the top-performing system, with TSA+VDP quite close and in the same group. Going now to the recall, TSA+BM25 is still the top-performing system, in the same group with TSA+VDP. SUMM is down below, once again, due to the redundancy in its answers.

SUMM presents the highest value of prec@1, thanks to its answer-building strategy, followed by TSA+VDP. In this case, all of the systems are in the same statistical group. For prec@5, TSA+VDP is the top-performing system thanks to its pruning strategy, which reduces the number of retrieved triples.

For BSBM 10M, TSA+BM25 is once again the system which obtains the greatest tb-DCG, in the same group of TSA+VDP. It is also the top-performing system concerning the recall. TSA+VDP is in the same statistical group of TSA+BM25, but the pruning heuristic and the limit of the parameter n significantly lowered the recall of the system. SUMM appears to be particularly ineffective with this measure.

For prec@1, TSA+VDP is the top-performing system, once again thanks to its pruning heuristic. The second system is TSA+BM25, and the third is SUMM. On this database, the strategy of SUMM does not seem to be enough to guarantee a high prec@1. Similar observations can be made for the prec@5.

Observing time, TSA+BM25 is the fastest algorithm on all three datasets. TSA+VDP follows. On LinkedMDB, TSA+VDP requires much more time than TSA+BM25, with an average of 425 seconds against 87. On the two synthetic databases, the time required is much more limited. Once again, SUMM proves to be the system that requires the highest average time, often because it meets the time limit of 1000s.

Finally, the required memory is often around tens of MB for all three systems. An exception to this is BSBM 10M, in the same way, BSBM 1M was an exception in its class of databases. The nature of this database is therefore confirmed even when its dimension scales.

TABLE 5. Report of the off-line times required by the different algorithms on the 10M datasets.

Database	algorithm	off-line time (min)
LinkedMDB	TSA	86
	SUMM	210
LUBM 10M	TSA	240
	SUMM	1168
BSBM 10M	TSA	259
	SUMM	1365

In Table 5, we report the off-line time required by the two systems TSA and SUMM over the 10M databases. As we can see, TSA behaves very well in all the databases. In the synthetic ones, it performs three times slower than on LinkedMDB 7M, but it still outperforms SUMM, which is four to five times slower than TSA.

This is due to the more efficient subgraph extraction and virtual-document creation strategy followed by the first algorithm.

C. BIG DATASETS (100M TRIPLES)

In this section we report the results of the experiments on the set of 100M datasets, comprising a version of DBPedia of 70M triples and the whole IMDB dataset of roughly 116M triples. TSA+BM25 and TSA+VDP are the only systems that scale to this size. Results are presented in Table 6.

Starting with IMDB 100M, TSA+BM25 performs poorly, while TSA+VDP performs much better. From this, we infer that the low performance of the first system is due primarily to the low quality of the ranking created by the BM25 method. Considering the recall we see that TSA+BM25 has 0.273, meaning that it retrieves relevant triples, but it is not very useful in the ranking phase. This is the same phenomenon seen in the experiments of Section IX-B. The dimension of the database has a significant impact on the final result, becoming more and more evident as the database grows.

Considering prec@1 and prec@5, the values are low for both systems. This is a consequence of the big dimension of the retrieved graphs as well as of the small number of relevant triples ranked at the top stemming from the high database level of "noise".

Let us consider DBpedia. TSA+VDP and TSA+BM25 feature similar tb-DCG values, with TSA+BM25 performing better than TSA+VDP in this case. In particular, the recall of TSA+BM25 is very high in this case (i.e., 0.851). Once

TABLE 6. Performances of the algorithms on the 100M databases. tb-DCG computed with $\lambda = 0$. The best system is in bold. † indicates the best statsystem through a t-test with $\alpha < 0.01$.

Dataset	Systems	tb-DCG	recall	prec@1	prec@5	time (sec)	memory (MB)
IMDB 100M	TSA+BM25	0.067±0.01	0.273±0.36	**0.011±0.00**	**0.009±0.00**	**36.42±01.10**	**2.60±00.10**
	TSA+VDP	**0.308±0.04**†	**0.363±0.04**	0.006±0.00	0.006±0.00	150.08±32.64	8.81±01.09
DBpedia	TSA+BM25	**0.135±0.01**	**0.851±0.03**†	0.000±0.00	0.000±0.00	**106.80±06.66**	**186.78±10.45**
	TSA+VDP	0.118±0.03	0.129±0.03	**0.004±0.00**	**0.002±0.00**	810.86±41.24	233.58±10.63

TABLE 7. Report of the off-line time required by the only algorithm able to scale TSA on the 100M databases.

Database	algorithm	off-line time (min)
IMDB 100M	TSA	3109
DBpedia	TSA	1242

again, this confirms the system's ability to extract subgraphs with relevant triples, but this also reflects its ranking limits.

In contrast, TSA+VDP obtains a low recall value, as the top n graphs of the BM25 ranking do not contain many relevant triples. In this particular circumstance, we also note that this low value of tb-DCG stems from the fact that the algorithm often exceeds the time-limit of 1000s, without finishing its execution. This is due to the high connectivity within the dataset, which requires more time for the creation, exploration, and pruning of the final answer graphs.

The vales of precision are quite low for both systems. The reason is the high number of noisy triples in the answers. TSA+BM25 obtains 0 for the values of precision but has a big recall. This means that the system can find many relevant triples, but cannot rank them adequately for the end-user. On the other hand, TSA+VDP cannot exploit the high recall produced by TSA+BM25, because the number of relevant triples contained in graphs goes beyond the n threshold.

Thus, if we consider the ability to retrieve relevant triples, DBPedia is handled quite easily by the TSA-based systems. On the other hand, the ranking phase requires some adjustments. In particular, it is recommended to improve the system's ability to reduce the level of noise within the answer graph.

In Table 7, we report the time required off-line for IMDB and DBPedia. The required time is much higher compared to the one required for the smaller databases. This is due to the dimension of the database. DBPedia requires a little less than one day, while IMDB 100M requires a little more than two days. It exceeds our time limit, even though we made an exception to obtain useful results. We estimated that the other system, SUMM, could not accomplish the task in less than eighteen days on both datasets, based on its progress after 48 hours. We, therefore, consider it unable to scale to these dimensions.

X. CONCLUSIONS

In this paper, we described the problem of keyword search over large RDF graphs. We proposed a system whose input is a query composed of just a few keywords, and the output is a ranking of connected subgraphs ranked by a similarity score with the query. To face it in a reproducible manner, we proposed a new evaluation methodology based on the concept of information need. We express an information need as a keyword and SPARQL query. The first is used in the typical best-match search scenario, while the second is employed in an exact-match search scenario. The keyword queries are the input of the search systems we have developed and tested; the SPARQL queries are evaluated to produce the Ground Truth – aka the correct answers to user information needs. Moreover, we developed a new evaluation framework based on three real databases: LinkedMDB, IMDB and DBPedia, and two synthetic databases: LUBM and BSBM. We considered datasets with different sizes, i.e., small (1M triples), medium (10M triples), and big (100M triples).

We designed and developed two effective, efficient, and scalable keyword search systems based on the virtual-document approach, called TSA+BM25 and TSA+VDP. We compared these two methods with other state-of-the-art keyword systems, namely: SLM, MRF-KS, and SUMM. In doing so, we extended and re-implemented MRF-KS to work on RDF graphs, providing a simplified implementation of SUMM, which guarantees the same performance in terms of effectiveness.

We evaluated the systems based on two criteria: efficiency (i.e., off-line and on-line execution time and required memory) and effectiveness (i.e., overall quality of the answers).

We showed that the TSA-based systems obtain the highest values of tb-DCG and recall over the small datasets. In particular, TSA+VDP is well-suited to work with real datasets, while TSA+BM25 is the best performing system with synthetic datasets. This was also confirmed with the medium size datasets. TSA+VDP is the top-performing system on LinkedMDB, while TSA+BM25 is the best system with synthetic databases. Considering the big datasets, with IMDB TSA+VDP is the top-performing system in terms of tb-DCG, while TSA+BM25 obtains the highest tb-DCG on DBpedia.

TSA-based systems have always proven to be a right choice in terms of tb-DCG and recall, obtaining good values in all the tested settings and scenarios. TSA+BM25, in particular, is always amongst the fastest systems, both off-line and on-line, while TSA+VDP introduces a trade-off between efficiency and effectiveness, even though it is faster than most of the state-of-the-art systems considered.

MRF-KS also obtains good values of tb-DCG on small real databases but does not scale to bigger databases. SLM features a graph-building strategy that often produces poor quality graphs and, consequently, a poor ranking. Moreover, performing most of its operations on-line, it does not scale to medium/big databases. The only other system that scales to medium datasets is SUMM that builds tree-shaped answers. In this way, it creates rankings that usually get a high precision; but it also has many overlapping, redundant triples in the answers, resulting in low values of tb-DCG. Moreover, the TSA-based systems always perform better than SUMM on the medium datasets. TSA+VDP obtains a higher tb-DCG on LinkedMDB, while TSA+BM25 always results as the top-performing systems in terms of tb-DCG and recall using synthetic datasets.

Overall, TSA-based systems have overcome the other systems limitations and have proved to be the top-performing systems or in the top-performing group in any scenario.

Regarding efficiency, TSA is up to *one order of magnitude* faster than MRF-KS in the off-line phase on small databases. It is usually in the same class of efficiency as SUMM. The only other system which is faster off-line is SLM, which only indexes the triples in the database. However, the on-line strategy of SLM is a real bottleneck of the system, which limits its ability to scale to bigger datasets. In the on-line phase, TSA+BM25 consistently proved to be one of the fastest systems in the statistically top-performing group.

We also showed how TSA-based systems scale to medium and big databases. Such results are promising in terms of future endeavors: TSA-based systems provide a solid ground to improve on-line efficiency.

As a future direction of research, we seek to investigate how the function that transforms a graph in a document can help the ranking performances function without damaging on-line efficiency. For example, one possible implementation could include words from the IRIs of the nodes and edges of the graph or the creation of fields within the virtual documents. Moreover, there is room to explore the best ways to efficiently produce the virtual documents on very large datasets (i.e., billions of triples). Another line of research would be to analyze how document-based systems work in a dynamic environment characterized by frequent updates (e.g., in the biomedical domain).

REFERENCES

[1] J. Pound, P. Mika, and H. Zaragoza, "Ad-hoc object retrieval in the Web of data," in *Proc. 19th Int. Conf. World Wide Web (WWW)*. New York, NY, USA: ACM, 2010, pp. 771–780.

[2] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *Proc. VLDB Endowment*, vol. 11, no. 4, pp. 420–431, 2017.

[3] W. Wu, "Proactive natural language search engine: Tapping into structured data on the Web," in *Proc. Joint EDBT/ICDT Conf.* New York, NY, USA: ACM, 2013, pp. 143–148.

[4] H. Bast, B. Buchhold, and E. Haussmann, "Semantic search on text and knowledge bases," *FNT Inf. Retr.*, vol. 10, no. 1, pp. 119–271, 2016.

[5] A. Kopliku, K. Pinel-Sauvagnat, and M. Boughanem, "Aggregated search: A new information retrieval paradigm," *ACM Comput. Surv.*, vol. 46, no. 3, pp. 1–31, Jan. 2014.

[6] J. X. Yu, L. Qin, and L. Chang, "Keyword search in relational databases: A survey," *IEEE Data Eng. Bull.*, vol. 33, no. 1, pp. 67–78, Mar. 2010.

[7] J. Coffman and A. C. Weaver, "An empirical performance evaluation of relational keyword search systems," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 30–42, 2014.

[8] H. Arnaout and S. Elbassuoni, "Effective searching of RDF knowledge graphs," *J. Web Semantics*, vol. 48, pp. 66–84, Jan. 2018.

[9] Q. Su and J. Widom, "Indexing relational database content offline for efficient keyword-based search," in *Proc. 9th Int. Database Eng. Appl. Symp. (IDEAS)*. Washington, DC, USA: IEEE Computer Society, Oct. 2006, pp. 297–306.

[10] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: Ranked keyword searches on graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*. New York, NY, USA: ACM, 2007, pp. 305–316.

[11] C. V. Gysel, M. De Rijke, and E. Kanoulas, "Neural vector spaces for unsupervised information retrieval," *ACM Trans. Inf. Syst.*, vol. 36, no. 4, pp. 1–25, Jun. 2018.

[12] J. Coffman and A. C. Weaver, "A framework for evaluating database keyword search strategies," in *Proc. 19th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*. New York, NY, USA: ACM, 2010, pp. 729–738.

[13] S. Elbassuoni and R. Blanco, "Keyword search over RDF graphs," in *Proc. 20th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*. New York, NY, USA: ACM, 2011, pp. 237–242.

[14] 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: CEUR-WS.Org, 2016.

[15] W. Le, F. Li, A. Kementsietsidis, and S. Duan, "Scalable keyword search on large RDF data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 11, pp. 2774–2788, Nov. 2014.

[16] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, and Y. Velegrakis, "QUEST: A keyword search system for relational data based on semantic and machine learning techniques," *Proc. VLDB Endowment*, vol. 6, no. 12, pp. 1222–1225, Aug. 2013.

[17] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proc. VLDB Endowment*, vol. 8, no. 1, pp. 73–84, Sep. 2014.

[18] J. A. Nasir, I. Varlamis, and S. Ishfaq, "A knowledge-based semantic framework for Query expansion," *Inf. Process. Manage.*, vol. 56, no. 5, pp. 1605–1617, Sep. 2019.

[19] D. Garigliotti, F. Hasibi, and K. Balog, "Identifying and exploiting target entity type information for ad hoc entity retrieval," *Inf. Retr. J.*, vol. 22, nos. 3–4, pp. 285–323, Aug. 2019.

[20] Z. Liu, C. Wang, and Y. Chen, "Keyword search on temporal graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 8, pp. 1667–1680, Aug. 2017.

[21] X. Lian, L. Chen, and Z. Huang, "Keyword search over probabilistic RDF graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 5, pp. 1246–1260, May 2015.

[22] A. Balmin, Y. Papakonstantinou, V. Hristidis, T. Wang, D. Srivastava, and N. Koudas, "A system for keyword proximity search on XML databases," in *Proc. VLDB Conf.* San Mateo, CA, USA: Morgan Kaufmann, 2003, pp. 1069–1072.

[23] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A system for keyword-based search over relational databases," in *Proc. 18th Int. Conf. Data Eng.*, Jun. 2003, pp. 5–16.

[24] Y. Luo, W. Wang, X. Lin, X. Zhou, J. Wang, and K. Li, "SPARK2: Top-k Keyword Query in relational databases," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 12, pp. 1763–1780, Dec. 2011.

[25] T. Tran, P. Cimiano, S. Rudolph, and R. Studer, "Ontology-based interpretation of keywords for semantic search," in *Proc. Semantic Web, 6th Int. Semantic Web Conf., 2nd Asian Semantic Web Conf. (ISWC ASWC)*, Busan, South Korea: Springer, Nov. 2007, pp. 523–536.

[26] J. Pound, A. K. Hudek, I. F. Ilyas, and G. Weddell, "Interpreting keyword queries over Web knowledge bases," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*. New York, NY, USA: ACM, 2012, pp. 305–314.

[27] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *Proc. 18th Int. Conf. Data Eng.* Washington, DC, USA: IEEE Computer Society, 2002, pp. 431–440.

[28] A. Simitsis, G. Koutrika, and Y. Ioannidis, "Précis: From unstructured keywords as queries to structured databases as answers," *VLDB J.*, vol. 17, no. 1, pp. 117–149, Nov. 2007.

[29] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *Proc. IEEE 23rd Int. Conf. Data Eng.* Washington, DC, USA: IEEE Computer Society, Apr. 2007, pp. 836–845.

[30] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, "STAR: Steiner-tree approximation in relationship graphs," in *Proc. IEEE 25th Int. Conf. Data Eng.* Washington, DC, USA: IEEE Computer Society, Mar. 2009, pp. 868–879.

[31] S. Bergamaschi, N. Ferro, F. Guerra, and G. Silvello, "Keyword-based search over databases: A roadmap for a reference architecture paired with an evaluation framework," *Trans. Comput. Collective Intell.*, vol. 21, pp. 1–20, 2016.

[32] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of SPARQL," *ACM Trans. Database Syst.*, vol. 34, no. 3, pp. 1–45, Aug. 2009.

[33] F. Picalausa and S. Vansummeren, "What are real SPARQL queries like?" in *Proc. Int. Workshop Semantic Web Inf. Manage. (SWIM)*. New York, NY, USA: ACM, 2011, pp. 7:1–7:6.

[34] A. Bonifati, W. Martens, and T. Timm, "An analytical study of large SPARQL Query logs," *Proc. VLDB Endowment*, vol. 11, no. 2, pp. 149–161, Oct. 2017.

[35] S. Büttcher, C. L. A. Clarke, and G. V. Cormack, *Information Retrieval: Implementing and Evaluating Search Engines*. Cambridge, MA, USA: MIT Press, 2010.

[36] D. Metzler and W. B. Croft, "A Markov random field model for term dependencies," in *Proc. 28th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*, 2005, pp. 472–479.

[37] D. Dosso, "Keyword search on RDF datasets," in *Proc. 41st Eur. Conf. IR Res. (ECIR)*. Springer, 2019, pp. 332–336.

[38] D. Dosso and G. Silvello, "A scalable virtual document-based keyword search system for RDF datasets," in *Proc. 42nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*. New York, NY, USA: ACM, 2019, pp. 965–968.

[39] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of IR techniques," *ACM Trans. Inf. Syst.*, vol. 20, no. 4, pp. 422–446, Oct. 2002.

[40] O. Hassanzadeh and M. P. Consens, "Linked movie data base," in *Proc. WWW Workshop Linked Data Web (LDOW)* vol. 538. Bolzano, Italy: CEUR-WS.Org, 2009.

[41] K. Balog and R. Neumayer, "A test collection for entity search in DBpedia," in *Proc. 36th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr. (SIGIR)*. New York, NY, USA: ACM, 2013, pp. 737–740.

[42] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A benchmark for OWL knowledge base systems," *J. Web Semantics*, vol. 3, nos. 2–3, pp. 158–182, Oct. 2005.

[43] C. Bizer and A. Schultz, "The berlin SPARQL benchmark," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2, pp. 1–24, 2009.

[44] C. Macdonald, R. McCreadie, R. L. T. Santos, and I. Ounis, "From puppy to maturity: Experiences in developing terrier," in *Proc. OSIR SIGIR*, 2012, pp. 60–63.

DENNIS DOSSO received the B.S. degree in computer science engineering and the M.S. degree in computer science engineering from the Department of Information Engineering, University of Padua, Italy, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree in information engineering.

His current research interests include keyword search and data citation.

GIANMARIA SILVELLO (Member, IEEE) is currently a Computer Engineer and an Assistant Professor (tenure-track) with the Department of Information Engineering, University of Padua. His research spans information retrieval, databases, digital libraries, and data citation. He has published over 120 scientific papers in national and international peer-reviewed venues.