

Keyword Search over Schema-less RDF Datasets by SPARQL Query Compilation

Yenier T. Izquierdo^a, Grettel M. García^a, Elisa Menendez^b,
Luiz André P. P. Leme^c, Angelo Neves^a, Melissa Lemos^a, Anna Carolina Finamore^d,
5 Carlos Oliveira^e, Marco A. Casanova^{a,*}

^a *Dept. Informatics and Tecgraf Institute, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro Brazil*

^b *Instituto Federal de Educação, Ciência e Tecnologia Baiano, Xique-Xique Brazil*

^c *Institute of Computing, Universidade Federal Fluminense, Niterói Brazil*

10 ^d *COPELABS, Universidade Lusófona, Lisboa, Portugal*

^e *ISEG-School of Economics and Management, Universidade de Lisboa; REM - Research in Economics and Mathematics, CEMAPRE, Lisboa Portugal*

Abstract

15 This article introduces an algorithm to automatically translate a user-specified keyword-based query K to a SPARQL query Q so that the answers Q returns are also answers for K . The algorithm does not rely on an RDF schema, but it synthesizes SPARQL queries by exploring the similarity between the property domains and ranges, and the class instance sets observed in the RDF dataset. It estimates set similarity based
20 on set synopses, which can be efficiently pre-computed in a single pass over the RDF dataset. The article includes two sets of experiments with an implementation of the algorithm. The first set of experiments shows that the implementation outperforms a baseline RDF keyword search tool that explores the RDF schema, while the second set of experiments indicate that the implementation performs better than the state-of-the-

* Corresponding author.

Email address: casanova@inf.puc-rio.br

URL: www.inf.puc-rio.br/~casanova

25 art TSA+BM25 and TSA+VDP keyword search systems over RDF datasets based on
the “virtual documents” approach.

Keywords: RDF keyword search; SPARQL; RDF; KMV-Synopses

1. Introduction

30 Keyword search is typically associated with information retrieval systems,
especially those designed for the Web. The user specifies a few terms, called keywords,
and it is up to the system to retrieve the documents, such as Web pages, that best match
the keywords. Keyword search over relational databases, as well as over RDF datasets,
has also been studied for some time. The challenge, in this case, is to retrieve the objects
35 that the keywords specify and to discover how they are interrelated. That is, an answer
for a keyword search over a relational database, or an RDF dataset, is not just a set of
objects, but a set of objects and relationships between them.

The keyword search tools proposed in the literature for the relational and the RDF
environments have points in common. However, RDF datasets pose an additional
challenge when no schema is defined, which is never the case for relational databases.
40 This is the primary motivation for the research reported in this article: keyword search
over schema-less RDF datasets.

Indeed, an RDF dataset does not require a predefined schema, that is, the user may
introduce new classes and properties, without defining them *a priori*, or changing the
current schema declaration if any. Hence, when compared to the relational model, RDF
45 offers the flexibility of schema-less datasets but, in this case, an RDF keyword search
tool must face the problem of retrieving sets of objects and their relationships without
resorting to schema information for guidance.

There are two other features of RDF that should be mentioned in the context of
keyword search. First, the adoption of RDF imposes no strict distinction between data
50 and metadata, that is, a keyword may match the name or description of a class or
property in the same way that it may match a data value. Second, an RDF dataset is
equivalent to a labeled graph, called an RDF graph, which allows addressing RDF

keyword search as a graph search problem. We will use the terms RDF dataset and RDF graph indistinctly in what follows.

55 In more detail, a keyword-based query is a set K of literals, or *keywords*. An answer for K over an RDF dataset T is a subset A of T such that: (i) A has triples that match some keywords in K ; and (ii) A induces a connected RDF graph. Note that we can then compare answers based on the number of keywords they match and on their number of triples, as defined in detail in [14]. Based on these remarks, we can informally state the
60 RDF keyword search problem (the RDF KwS-Problem) as: “Given an RDF dataset T and a keyword-based query K , find an answer A for K over T , preferably with as many keyword matches as possible and with the smallest set of triples as possible”.

Let G_A be the RDF graph induced by an answer A . If G_A is a Steiner tree of T that covers the nodes that match keywords, then G_A is connected and does not have
65 unnecessary edges. Therefore, a basic strategy to solve the RDF KwS-Problem would be to construct an algorithm that tries to find as many keyword matches as possible and, at the same time, find a Steiner tree of T that covers the matching nodes, which is a challenging task.

The central contribution of this article is an algorithm to address the RDF KwS-
70 Problem by automatically translating a keyword-based query K into a SPARQL query Φ so that the answers Φ returns are also answers for K . The novelty of the algorithm lies in that it neither relies on an RDF schema, nor accesses the RDF graph during the compilation process – and this is a relevant feature, but it synthesizes SPARQL queries by exploring the similarity between the property domains and ranges and the class
75 instance sets observed in the RDF dataset. To achieve good performance, even for large RDF datasets, the algorithm, in turn, estimates set similarity based on KMV-synopses [7]. The KMV-synopses are pre-computed efficiently in a single pass over the RDF dataset and stored together with the RDF dataset for later use by the compilation process.

80 The second contribution consists of two sets of comprehensive experiments with an implementation of the algorithm. The first set of experiments shows that the implementation outperforms, in all metrics adopted, a baseline RDF keyword search tool that explores the RDF schema. These results suggest that schema information can

indeed be replaced by pre-computed, concise K_{MV}-synopses for the property domains
85 and ranges, and class instance sets. These results are, to some extent, unexpected, since
the lack of schema information seemed difficult to overcome from the onset. The
second set of experiments indicate that the implementation performs better than the
TSA+BM25 and TSA+VDP keyword search systems over RDF datasets based on the
“virtual documents” approach, using the metrics and the benchmarks proposed
90 originally to assess these systems.

As a third contribution, we propose the *Graph Relevance Ratio (GRR)* to establish
when an answer graph is relevant w.r.t. a ground truth graph. It is based on the number
of relevant and non-relevant triples in the RDF graph, but it punishes the presence of
non-relevant triples, and does not memorize the relevant triples in previous rank
95 positions.

The remainder of this article is organized as follows. Section 2 summarizes related
work. Section 3 provides the necessary background. Section 4 describes the proposed
algorithm to compile keyword-based queries into SPARQL queries. Section 5 covers
the first set of experiments that compare an implementation of the proposed algorithm
100 with a baseline RDF keyword search tool that explores the RDF schema. Section 6
describes the second set of experiments that compare the implementation with the state-
of-the-art TSA+BM25 and TSA+VDP keyword search systems over RDF datasets
based on the “virtual documents” approach. Finally, Section 7 contains the conclusions
and suggests directions for future research.

105 **2. Related Work**

A survey of keyword-based query processing tools over relational databases and
RDF datasets can be found in [4].

Early relational keyword-based query processing tools [1][2][21][22][35] explored
the foreign/primary keys declared in the relational schema to compile a keyword-based
110 query into an SQL query with a minimal set of join clauses – and this is a key idea –
based on the notion of candidate networks (CNs). This approach was also adopted in
recent tools [6][35]. In particular, QUEST [6] explores the structure of the conceptual

schema to synthesize an SQL query based on a Steiner tree that induces a minimum set of joins. A recent article [41] specifically investigated CN scoring functions and empirically demonstrated that the proposed function outperforms earlier scoring functions.

A recent article [36] also explored relational schema information to compile keywords into SQL queries over databases exposed on the Web. The system the authors proposed, called SQUIRREL, selects and ranks relational databases on the Web, based on the metadata the databases expose, pre-processes the keywords, which includes identifying aggregation functions the keywords might express, and compiles the pre-processed keywords into SQL queries. The authors compared favorably SQUIRREL with an earlier system [5] on a relational database with 9 keyword queries. This comparison was extended in [34].

The algorithm proposed in this article compiles a keyword-based query into a SPARQL query that includes restriction clauses that represent keyword matches and join clauses that connect the restriction clauses. Each answer of the SPARQL query then corresponds to a subgraph of the RDF graph that contains literal nodes that match the keywords and paths that connect the literal nodes. Without such join clauses, an answer would be a disconnected set of nodes of the RDF graph, which hardly makes sense. The generation of the join clauses builds upon the idea of candidate networks.

An RDF keyword-based query processing tool can be *schema-based*, when it exploits the RDF schema to compile a keyword-based query into a SPARQL query, or *graph-based*, when it directly explores the RDF dataset or summaries thereof. The algorithm described in this article falls into this last category.

QUIOW [14][23] is a fully automatic, schema-based tool that supports keyword-based query processing for both the relational and RDF environments. The tool constructs a Steiner tree that covers a set of nodes (relation schemes or RDF classes) whose instances match the largest set of keywords and incorporates a backtracking step to further expand the keyword-query results by generating alternative (SQL or SPARQL) queries. The experiments reported in this article adopt this tool as a baseline.

QUICK [46] is another example of an RDF schema-based tool. It translates keyword-based queries to SPARQL queries with the help of the user, who chooses a set of intermediate queries, which the tool ranks and executes.

145 As for graph-based tools, SPARK [48] uses techniques, such as synonyms from Wordnet and string metrics, to map keywords to knowledge base elements. The matched elements in the knowledge base are then connected by minimum spanning trees from which SPARQL queries are generated. A recent article [37] also explores WordNet and proposes a ranking method to implement keyword search over RDF
150 graphs. Elbassuoni and Blanco [13] described a technique to retrieve a set of subgraphs that match the keywords and to rank them based on statistical language models. Ranking answers of a keyword-based query is addressed, for example, in [15]. Keyword expansion, as reported in [36][37][48], and the processing of reserved terms, as is [23][36], are complementary to the discussion in this article.

155 Han et al. [19] described an algorithm that uses the keywords to first obtain elementary query graph building blocks, such as entity/class vertices and predicate edges, and then applies a bipartite graph matching-based best-first search method to assemble the final query. Tran et al. [38] introduced the idea of generating summary graphs for the RDF graph, using the class hierarchy, to generate and rank candidate
160 SPARQL queries. Le et al. [25] also proposed to process keyword queries using another RDF graph summarization algorithm. Zheng et al. [47] adopted a pattern-based approach. Lin et al. [27] summarized all the inter-entity relationships from RDF data to translate keywords to SPARQL queries. Finally, Wen et al. [43] introduced another graph summarization technique that amounts to recovering an RDF schema from the
165 RDF graph. We abandoned a similar strategy early on, in preliminary experiments, since the algorithm proposed in this article outperformed it.

Wang et al. (2008) [42] proposed a clustered-graph structure that summarizes the original ontology. This reduced data space is then used to compute top-k queries, which are ranked by query length, the relevance of ontology elements w.r.t. the query and the
170 importance of ontology elements. The authors use TAP, DBLP and LUBM for the experiments, and introduce a new metric, called Target Query Position (TQP). Section

4.4 explain how we rank the query results, which also depends on the importance of the RDF nodes, classes and properties. LUBM is one of the datasets we use in Section 6.2.

175 Gkirtzou et al. (2015) [16] presented an approach for keyword search for temporal RDF graphs that automatically compiles keyword queries into a set of candidate SPARQL queries. To support temporal exploration, the method is enriched with temporal operators allowing the user to explore data within predefined time ranges. The novelty of the approach lies exactly on this enrichment, which we do not explore in this article. In particular, the authors use the reciprocal rank metric in the experiments, as
180 we do in Section 5.2.

Yoghourdjian et al. (2017) [45] developed a retrieval model for keyword queries over RDF knowledge graphs that only retrieves the top-k scored subgraphs for the given query based on a scoring function. The authors adopted YAGO a large-scale general-purpose RDF knowledge graph derived from Wikipedia and WordNet, and the average
185 NDCG as metric for the experiments. The query compilation strategy described in Section 4.3 also includes a ranking score, described in Section 4.4, that limits the query results to the top-k. Sections 5.2 and 6.2 show that the combination of the query compilation approach based on KMV-synopses and the ranking function outperforms state-of-the-art approaches, include an approach based on schema information.

190 Ma et al. (2018) [28] described a keywords-to-SPARQL translation process that circumvents the lack of underlying schema information. They compute, from the RDF data graph, an inter-entity relationship summary with complete schema information, and adopt a search prioritization scheme that combines the degree of a vertex with the distance from the original keyword element. Finally, the proposed approach finds the
195 top-k subgraphs, which are relevant to the conjunction of the entering keywords. The approach we propose in this article uses KMV-synopses to capture graph information, as explained in Section 4.3, and ranks the SPARQL query results based on a more sophisticated node importance measure, as discussed in Section 4.4.

200 Dosso and Silvello (2020) [12] proposed the TSA+BM25 and the TSA+VDP keyword search systems over RDF datasets based on the “virtual documents” approach. These systems move 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. The authors show that these approaches are more efficient and effective,

when compared with state-of-the-art systems. The paper also describes a benchmark
205 that contains three real datasets, LinkedMDB, IMDb and a subset of DBpedia, as
defined in [3], and two synthetic databases, the Lehigh University Benchmark (LUBM)
[17] and the Berlin SPARQL Benchmark (BSBM) [8]. The benchmark and the code
required to run the experiments are publicly available. Section 6 compares the approach
proposed in this article with these keyword search systems, using IMDb, a subset of
210 DBpedia, and the 10M versions of LUBM and BSBM.

Contrasting with these approaches, the algorithm described in this article adopts
KMV-synopses [7] to concisely represent the property domains and ranges and class
instance sets. The algorithm then uses the KMV-synopses to estimate set similarity
measures (see Section 3) that, in turn, drive the process of compiling a keyword-based
215 query into a SPARQL query (see Section 4.3). The algorithm also incorporates RDF
resource ranking [29] to improve the query compilation process and to rank answers
(see Section 4.4).

KMV-synopses [7] permit estimating the cardinality of multiset expressions and
several set similarity measures, including a generalized Jaccard similarity measure for
220 more than two sets. Hadjieleftheriou et al. [18] used set synopses to estimate the size of
the result of set similarity queries. Yang et al. [44] introduced a KMV sketch technique
to address the problem of approximating containment similarity search. Venetis et al.
[40] proposed a similarity index for set-valued features based on KMV-synopses. The
index methods proposed in these last two references could be useful to filter the
225 candidate property domains and ranges and class instance sets to be included in a
SPARQL query during the query compilation process. Finally, Le et al. [26] chose to
use KMV-synopses precisely because they allow estimating the size of the intersection
of multiple sets (not just binary intersection) in the context of rewriting queries on
SPARQL views. Their motivation for adopting KMV-synopses is, therefore, quite
230 similar to ours. Le et al. also briefly commented on the problem of KMV-synopses
maintenance, which is outside the scope of this article. The present implementation
recomputes the KMV-synopses when necessary, much in the same way that database
systems recompute statistics.

We now comment on the benchmarks we adopted. In Section 5, we adopted a
235 variation of the Coffman and Weaver benchmark [10]. Originally, Coffman and Weaver
[10] presented a qualitative evaluation of several relational keyword-based query
processing tools, using a benchmark that consists of a simplified version of IMDb, a
subset of Wikipedia, and a subset of the Mondial database, and a set of 50 keyword
queries for each database. For each keyword query, the authors manually defined an
240 equivalent SQL query and judged the relevance of the results returned according to the
keyword query information needs. As briefly detailed in Section 5, the RDF keyword
search benchmark adopted in this article adapts Coffman’s benchmark in several
significant aspects, including avoiding the manual analysis of query results. A recent
benchmark for relational keyword-based systems also proposed to use Mondial and
245 IMDb, as well as DBLP and Northwind [34].

In Section 6, we adopt, in part, the benchmark proposed in [12]. We used a second
version of IMDb, a subset of DBpedia, as defined in [3], the Lehigh University
Benchmark (LUBM) [17] and the Berlin SPARQL Benchmark (BSBM) [8].

For IMDb, Dosso and Silvello designed 50 keyword queries, together with their
250 correct translations to SPARQL queries. For DBpedia, the authors considered 50 topics
from the classes QALD2_te and QALD2_tr used by [3].

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
255 queries. Dosso and Silvello converted these queries to construct queries and produced
their equivalent keyword query. BSBM is a database built on an e-commerce use case,
where different vendors with posted reviews offer a set of products. Dosso and Silvello
selected the Explore use case with 13 different SPARQL queries, which they converted
to construct SPARQL queries and keyword queries, as for LUBM. In Section 6.2, we
260 adopted the 10 million triples versions of these datasets.

Finally, for the purposes of our comparison, described in Section 6, we considered
it unnecessary to repeat the experiments for LinkedMDB, since this dataset is much
smaller than the triplified full IMDb, which we used. We also discarded the smaller,
1M triple versions of LUBM and BSDM.

265 3. Background

3.1 RDF

An *Internationalized Resource Identifier (IRI)* is a global identifier that denotes a *resource*. RDF describes data as *triples* of the form (s,p,o) , where s is the *subject*, p is the *predicate* or *property*, and o is the *object* of the triple [11]. The subject of a triple is an IRI or a blank node, the property is an IRI, and the object is an IRI, a blank node, or a literal. An *RDF dataset* is a set T of RDF triples; T is equivalent to a labeled graph G_T whose set of nodes is the set of RDF terms that occur as subject or object of the triples in T , and there is an edge (s,o) in G_T labeled with p iff $(s,p,o) \in T$.

We may directly scan an RDF dataset T to discover the *observed* classes and properties, the property domains and ranges, and the set of class instances, defined as follows. A property p is *observed* in T iff there is a triple of the form (s,p,o) in T . The *observed domain* of p in T , denoted D_p , refers to the set of IRIs s such that $(s,p,o) \in T$, the *observed range* of p to the set of IRIs or literals o such that $(s,p,o) \in T$, the *observed non-literal range* of p , denoted R_p , to the set of IRIs in the range of p , and the *observed set of instances* of a class c (or the *observed class instances*), denoted S_c , to the set of IRIs s such that $(s, \text{rdf:type}, c) \in T$. From this point on, we will omit the qualifier *observed* whenever possible. We will also use as synonyms the terms instance and entity.

RDF schema extends the vocabulary of RDF to define classes and properties and hierarchies thereof [9]. SPARQL 1.1 is the standard query language to access RDF datasets [20]. We will introduce the details of the language along the text, as needed.

3.2 Keyword-based Queries

Let T be an RDF dataset. A *keyword* is an RDF literal. A *keyword-based query* is a finite set $K = \{k_1, \dots, k_n\}$ of keywords.

Let \mathcal{L} be the set of all literals. A *match function* $\mu: \mathcal{L} \times \mathcal{L} \rightarrow \text{Bool}$ maps each pair of literals into a Boolean value and is such that $\mu(L_1, L_1) = \text{True}$ and $\mu(L_1, L_2) = \mu(L_2, L_1)$, for

any $L_1, L_2 \in \mathcal{L}$. We say that L_1 and L_2 *match* iff $\mu(L_1, L_2) = \text{True}$. We say that a triple $(s, p, o) \in T$ *matches* a literal L iff o is a literal and L and o match; we also say that o is a *matching node* of G_T .

An *answer* for K over T is a subset A of T such that there is $K_A \subseteq K$, the set of *matched keywords*, and $A_K \subseteq A$, the set of *triple matches*, such that:

- for each $k \in K_A$, there is $(s, p, o) \in A_K$ that matches k
- for each $(s, p, o) \in A_K$, there is $k \in K_A$ matched by o
- The RDF graph G_A induced by A is connected

As already observed in the introduction, we can compare answers based on the number of keywords they match and on their number of triples. Furthermore, if G_A is a Steiner tree of T that covers the matching nodes, then G_A is connected and does not have unnecessary edges. Therefore, a basic strategy to solve the RDF KwS-Problem would be to construct an algorithm that tries to find as many keyword matches as possible and, at the same time, find a Steiner tree of T that covers the matching nodes. If the algorithm cannot find one such Steiner tree, it must abandon some of the keyword matches and restart the search for a Steiner tree.

3.3 Set Similarity Measures and KMV-Synopses

As mentioned in the introduction, the novelty of the algorithm lies in that it synthesizes SPARQL queries by exploring the similarity between the property domains and ranges and the class instance sets observed in the RDF dataset. To achieve good performance, the algorithm estimates set similarity based on KMV-synopses [7]. This section summarizes the essentials of these two aspects: set similarity and KMV-synopses.

Let \mathcal{D} be the universe and let $A_1, \dots, A_n \subseteq \mathcal{D}$.

The *Jaccard similarity measure* is a well-known way to estimate the similarity of two or more sets, A_1, \dots, A_n , based on what elements they have in common, without giving preference to any of the sets; the measure is normalized by the number of elements in the union of the sets. The *set containment similarity measure* is adopted

when one wants to find, given a set A_i , which other sets A_j are similar to A_i , based on
 320 the number of elements that A_i and A_j have in common; the measure is normalized by
 the number of elements in A_i .

More precisely, the n -way Jaccard similarity measure of A_1, \dots, A_n is defined as:

$$J(A_1, \dots, A_n) = \frac{|A_1 \cap \dots \cap A_n|}{|A_1 \cup \dots \cup A_n|} \text{ if } A_1 \cup \dots \cup A_n \neq \emptyset \quad (1)$$

$$J(A_1, \dots, A_n) = 1 \quad \text{otherwise} \quad (2)$$

and the set containment similarity measure of A_i and A_j as:

$$C(A_i, A_j) = \frac{|A_i \cap A_j|}{|A_i|} \quad \text{if } A_i \neq \emptyset \quad (3)$$

$$C(A_i, A_j) = 1 \quad \text{otherwise} \quad (4)$$

Let k be a positive integer. Intuitively, a *KMV-synopsis* of a set $S \subseteq \mathcal{D}$ defines a
 325 random sample of S of size k , with the help of a hash function. Using the *KMV-synopsis*
 one can then estimate the cardinality of S . Furthermore – and this is the important
 property for this article – one can estimate the Jaccard similarity and the set containment
 of the sets involved using their *KMV-synopses*.

More precisely, let h be a hash function from \mathcal{D} to $\{0, \dots, M\}$ (with $M \sim O(|\mathcal{D}|^2)$). The
 330 *KMV-synopsis* of a set $S \subseteq \mathcal{D}$ is the set V of the k smallest values of the set
 $\{v \in \{0, \dots, M\} \mid v = h(s) \text{ and } s \in S\}$. If h is a perfect hash function, then V induces a
 random sample $W = \{s \in S \mid h(s) \in V\}$ of S of size k .

Let $U_{(k)}$ denote the k^{th} smallest value of the *KMV-synopsis* V , divided by M . Then,
 an estimation $\overline{|S|}$ of $|S|$ is

$$\overline{|S|} = \frac{(k-1)}{U_{(k)}} \quad (5)$$

335 with absolute ratio error given by [7]:

$$E \left[\frac{\text{abs}(\overline{|S|} - |S|)}{|S|} \right] \approx \sqrt{\frac{2}{\pi(k-2)}} \quad (6)$$

Let V_1, \dots, V_n be the *KMV-synopses* of A_1, \dots, A_n , k_1, \dots, k_n be the sizes of V_1, \dots, V_n ,
 and $k = \min(k_1, \dots, k_n)$. Then, we define $V = V_1 \oplus \dots \oplus V_n$ as the set of the k smallest
 values in $V_1 \cup \dots \cup V_n$. Let $U_{(k)}$ denote the k^{th} smallest value of $V = V_1 \oplus \dots \oplus V_n$, divided
 by M , as before. Finally, let $K_\cap = |V_1 \cap \dots \cap V_n|$. Then, the following estimations hold

340 [7]:

$$\overline{|A_1 \cup \dots \cup A_n|} = \frac{(k-1)}{U_{(k)}} \quad (7)$$

$$\overline{J(A_1, \dots, A_n)} = \frac{K_{\cap}}{k} \quad (8)$$

$$\overline{|A_1 \cap \dots \cap A_n|} = \frac{K_{\cap}(k-1)}{k U_{(k)}} \quad (9)$$

$$\overline{C(A_i, A_j)} = \frac{|A_i \cap A_j|}{|A_i|} \quad (10)$$

A detailed analysis of the accuracy of KMV-synopses to estimate unions, intersections, and Jaccard distance can be found in [7]. Section 2 already pointed out other references that successfully adopt KMV-synopses for a variety of query optimization problems [18][26][33][36].

345 To illustrate this point, consider the IMDb version adopted in [12]. Table 1 shows the Jaccard values, as directly computed from the dataset, and the estimations obtained using KMV-synopses, with $K=8,192$ and $K=32,768$. Note that the accuracy of the Jaccard estimations obtained using KMV-synopses, with $K=8,192$, are quite reasonable.

350 **Table 1 – An example of approximating the Jaccard similarity measure using KMV-synopses of different sizes.**

Set A	Set B	$J(A, B)$	$K = 8,192$			$K = 32,768$		
			$\overline{J(A, B)}$	Abs diff.	Error vs Exact	$\overline{J(A, B)}$	Abs diff.	Error vs Exact
imdb:name	D_imdb:know_for_titles	0.8237	0.8253	0.0016	0.19%	0.8243	0.0006	0.07%
imdb:title	R_imdb:know_for_titles	0.2205	0.2185	0.0020	0.91%	0.2184	0.0021	0.95%
imdb:name	R_imdb:principal_cast	0.1768	0.1860	0.0092	5.20%	0.1856	0.0088	4.98%
imdb:title	D_imdb:principal_cast	0.1095	0.1057	0.0038	3.47%	0.1061	0.0034	3.11%
imdb:name	R_imdb:directors	0.0614	0.0593	0.0020	3.30%	0.0599	0.00145	2.36%
imdb:title	D_imdb:directors	0.5838	0.5844	0.0005	0.09%	0.5839	9.2E-05	0.02%
imdb:name	R_imdb:writers	0.0775	0.0747	0.0027	3.54%	0.0755	0.001952	2.52%
imdb:title	D_imdb:writers	0.5114	0.5067	0.0046	0.91%	0.5088	0.00255	0.50%

Notes:

1. imdb:title and imdb:name represent the sets of synopses of classes Title and Name, respectively.
 2. D_p indicates the domain of property p and R_p indicates the range.
 3. $J(A, B)$ indicates the Jaccard similarity measure between sets A and B.
- 355

4. Compiling Keyword-based Queries into SPARQL

4.1 A Motivating Example

This section describes a simplified example that illustrates how the proposed algorithm translates a keyword-based query K into a SPARQL query Φ so that the answers Φ returns are also answers for K .

Example 1: Let T be the RDF dataset whose graph is shown in Figure 1. In what follows, let D_p and R_p denote the observed domain and the observed range of a property p observed in T .

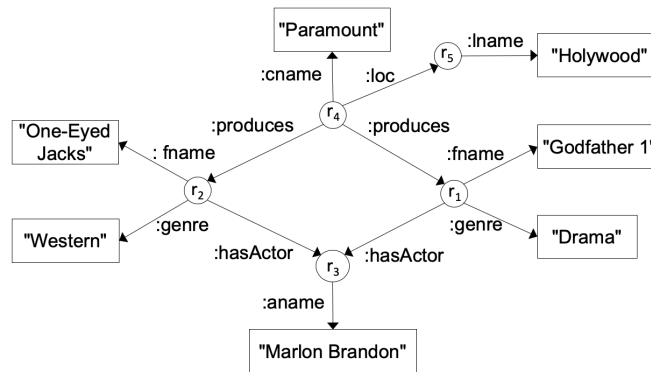


Figure 1. The graph G of an RDF dataset T .

Pre-processing. Before processing any keyword-based query, the algorithm executes a single scan of T that simultaneously pre-computes KMV-synopses for the observed property domains, non-literal ranges, and class instance sets. The KMV-synopses are stored with the RDF dataset to be later used to estimate set similarity. During the scan, the algorithm also pre-computes indexes for T that, given a keyword k , return the names of the properties that have values that match k .

Consider the keyword-based query

$$K = \{One-Eyed, Western, Brandon, Hollywood\} \quad (11)$$

The algorithm uses a data structure, called a *query forest*, as in Figures 2 to 6, where a rectangular node is labeled with a keyword and an oval node with a list of property domains and ranges, to be interpreted as indicating their intersection. The algorithm starts with a query forest as in Figure 2 and then gradually tries to reduce the forest to a single tree by using three operations – *node fusion*, *edge addition*, and *tree expansion*. Lastly, it compiles the final forest into a SPARQL query that returns answers for K .

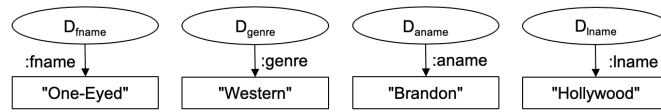


Figure 2. Initial query forest for the keyword-based query

$$K = \{One-Eyed, Western, Brandon, Hollywood\}.$$

Keyword matching. First observe that the query in (11) identifies entity sets by listing keywords that should match property values of the entities. So, the first step is to match the keywords with property values and identify which are these properties. We admit partial matches so that, for example, the keyword “*One-Eyed*” matches the literal “*One-Eyed Jack*”. Assume that the indexes return the following matches for K :

Western matches some value of $:genre$ (12)

One-Eyed matches some value of $:fname$ (13)

Brandon matches some value of $:aname$ (14)

Hollywood matches some value of $:lname$ (15)

This information is represented as an initial query forest, as shown in Figure 2.

Consider the leftmost tree. The oval node is labelled with D_{fname} and the rectangular node with “*One-Eyed*” to indicate that there is at least one entity e in the domain of $:fname$ such the value of $:fname$ for e is a string that matches “*One-Eyed*”. The last step of the process synthesizes a SPARQL query that locates the set of all such entities. The other trees in Figure 2 should be likewise interpreted.

If a keyword matches values of several properties, one such match is chosen, using a combination of a literal matching score and a node ranking score. We refer the reader to Section 4.4 for a brief description of the disambiguation strategy adopted.

395 *Node fusion*. The entities in a set might be identified by more than one property value,
 that is, by more than one keyword. In general, the algorithm handles this situation
 inspecting only the KMV-synopses of the property domains, through an operation
 400 called *node fusion*. In terms of a query forest, node fusion combines two trees by finding
 two nodes, one from each tree, that can be profitably replaced by a single node. By
 profitable we mean that the sets that label the nodes to be fused have a high Jaccard
 similarity value. Note that the Jaccard similarity is computed over a list of 2 or more
 405 sets, as in Eq. (1). This indicates that, with a high probability, one may find entities s
 such that the properties that label the nodes are all defined for s (see Prop. 1a). Again,
 the last step of the process synthesizes a SPARQL query that takes this situation into
 account.

Figure 3 shows the query forest after the fusion of the roots of the two leftmost trees,
 405 respectively labeled with D_{fname} and D_{genre} .

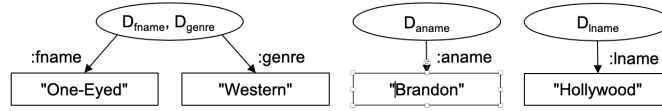


Figure 3. The query forest after node fusion.

This decision is justified by estimating the Jaccard similarity between each pair of
 sets that label the roots of the trees in Figure 2. For this very simple example, we
 computed the Jaccard similarities by just observing the RDF graph in Figure 1, that is,
 we in fact ignore the KMV-synopses since all sets involved in this example have low
 410 cardinality. Since $D_{fname} = D_{genre} = \{r_1, r_2\}$, we have:

$$J(D_{fname}, D_{genre}) = 1 \quad (16)$$

which implies that any resource s drawn from $D_{fname} \cup D_{genre}$ is in $D_{fname} \cap D_{genre}$.
 Furthermore, any other pair of sets that label the roots of the trees in Figure 2 are
 disjoint. Hence, their Jaccard similarity is 0. Thus, any other pair of roots are not good
 candidates for node fusion.

415 A new node, labeled with $\{D_{fname}, D_{genre}\}$, replaces the original nodes to signal that
 now we want to find entities that are in the intersection $D_{fname} \cap D_{genre}$ (each such entity
 has values for both properties `:fname` and `:genre` that may match two keywords, “*One-
 Eyed*” and “*Western*”).

The entity sets identified by the keywords do not constitute answers, though, since
 420 an answer to a keyword query has to indicate how the entities are related. The algorithm
 then tries to identify what paths might exist in the RDF graph that connect the entities,
 using only the KMV-synopses, without actually traversing the graph. This is purpose
 of two other operations, called *edge addition* and *tree expansion*.

425 *Edge addition.* Edge addition tries to find an object property p that might directly
 connect two entities. In the example, the key point is that the domain of property `:title`
 and the domain of property `:genre` might have elements in common with the domain of
 the object property `:hasActor`; simultaneously the range of `:hasActor` might have elements
 in common with the domain of `:aname` (*actor name*). This is detected again using set
 430 similarity, estimated using the pre-computed KMV-synopses. The last step of the
 process synthesizes a SPARQL query that has a join clause that takes this situation into
 account.

In terms of a query forest, edge addition tries to combine two trees by finding two
 nodes, a_1 and a_2 , one from each tree, that can be profitably connected by a new edge,
 435 labeled with a property p , in the following sense. Let S_i be the intersection of the sets
 that label a_i . One should select nodes a_1 , a_2 , and a property p so that, with a high
 probability, there is a triple (s,p,o) in T such that s is in the intersection of S_1 and the
 domain of p , given that we know that s is in S_1 , and o is in the intersection of S_2 and the
 range of p , given that we know that o is in S_2 (see Prop. 1d). We use set containment
 440 for this purpose, as explained below.

Figure 4 shows the query forest after combining the two leftmost trees in Figure 3
 by adding an edge, labeled with “`:hasActor`”.

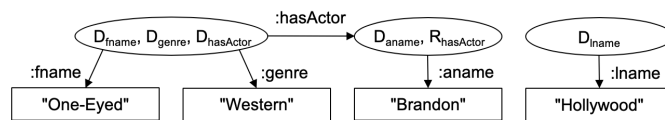


Figure 4. The query forest after adding an edge labeled with `:hasActor`.

This decision is based on the fact that the set containment of $D_{fname} \cap D_{genre}$ and
 $D_{hasActor}$ is

$$C(D_{fname} \cap D_{genre}, D_{hasActor}) = 1 \quad (17)$$

445 which implies that any resource s in $D_{fname} \cap D_{genre}$ is also in $(D_{fname} \cap D_{genre}) \cap D_{hasActor}$. Note that we switched from Jaccard to set containment since we now know that s is in $D_{fname} \cap D_{genre}$ and we want to find a property p that maximizes the chances that s is also in D_p .

450 But this is not enough since the new edge should connect the two nodes. By a similar argument, the set containment similarity of D_{aname} and $R_{hasActor}$ is

$$C(D_{aname}, R_{hasActor}) = 1 \quad (18)$$

which implies that any resource o in D_{aname} is also in $D_{aname} \cap R_{hasActor}$. Putting the two arguments together, we found a property, `:hasActor`, that maximizes the chances that s is in its domain and o is in its range. That is, the choice of `:hasActor` maximizes the product (see Proposition 1d):

$$C(D_{fname} \cap D_{genre}, D_{hasActor}) \times C(D_{aname}, R_{hasActor}) \quad (19)$$

455

Tree expansion. Tree expansion is a relaxation of edge addition in the sense that it does not require that both the domain and range of an object property be similar to other sets already under consideration; it suffices to have just the domain or just the range. The repeated application of tree expansion, combined with edge addition, tries to find longer paths to connect entities in the sets of already identified. For example, the range of the object property `:loc` (*location*) is similar to the domain of the property `:lname` (*location name*), so it might be profitable to combine the two properties into a path of length 2. Once again, this is detected using set similarity, estimated using the pre-computed KMV-synopses, and the last step of the process synthesizes a SPARQL query that has a join clause that takes this situation into account.

460

465

In terms of a query forest, tree expansion adds a node and an edge to create a new forest that might be transformed in a later step by node fusion or edge addition. The choice of which edge to include is similar to edge addition, except that one of the nodes is new, that is, added together with the edge. Tree expansion is used when node fusion and edge addition cannot be applied and is the last resort of the method. This point is better explained at the end of Section 4.3, when trees expansion is covered in detail.

470

For example, it follows from Figure 1 that D_{lname} is disjoint from the domain or range of any property, except for the range of `:loc`. Hence, we have:

$$J(D_{fname}, D_{genre}, D_{hasActor}, D_{lname}) = 0 \quad (20)$$

$$J(D_{aname}, R_{hasActor}, D_{lname}) = 0 \quad (21)$$

$$C(D_{fname} \cap D_{genre} \cap D_{hasActor}, D_{lname}) = 0 \quad (22)$$

$$C(D_{aname} \cap R_{hasActor}, D_{lname}) = 0 \quad (23)$$

475 which implies that we cannot use node fusion or edge addition, as in the previous steps, to create a single tree out of the forest in Figure 4. Tree expansion, therefore, adds a new node, labeled with D_{loc} , and a new edge, labeled with “:loc”, creating the forest shown in Figure 5.

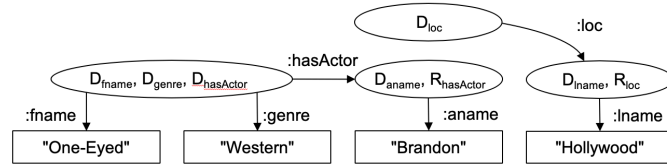


Figure 5. The query forest after expanding the rightmost tree by adding an edge labeled with “:loc”.

Edge addition. To conclude the construction of the query forest for the running example, it follows from Figure 1 that

$$C(D_{fname} \cap D_{genre} \cap D_{hasActor}, R_{produces}) = 1 \quad (24)$$

$$C(D_{loc}, D_{produces}) = 1 \quad (25)$$

480 As before, this suggests that we can add an edge, labeled with “:produces”, to combine the two trees in Figure 5, creating the final query tree, shown in Figure 6.

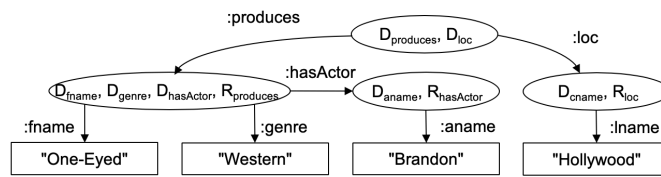


Figure 6. The final query tree after adding an edge labeled with “:produces”.

SPARQL query compilation. The last step is to generate a SPARQL query Φ . In this simple example, the final forest consists of a single tree. When this is not the case, the tree with the largest number of keyword matches is kept.

485 In terms of a query forest, for each node of the tree, the WHERE clause of Φ has a variable and, for each edge, a join clause or a FILTER clause. The construction of the forest and the generation of Φ also guarantees that any answer A that Φ returns is such that G_A is a Steiner tree.

The WHERE clause of the SPARQL query generated from the final query tree in
490 Figure 6 is:

1. $?v_1$:fname $?v_2$ FILTER (*match*($?v_2$, “One-Eyed”)) .
2. $?v_1$:genre $?v_3$ FILTER (*match*($?v_3$, “Western”)) .
3. $?v_4$:aname $?v_5$ FILTER (*match*($?v_5$, “Brandon”)) .
4. $?v_6$:lname $?v_7$ FILTER (*match*($?v_7$, “Hollywood”)) .
5. $?v_1$:hasActor $?v_4$.
6. $?v_8$:loc $?v_6$.
7. $?v_8$:produces $?v_1$

Observing Figure 6, the leftmost oval node, a_1 , corresponds to the variable “ $?v_1$ ”; the two edges from a_1 to rectangular nodes correspond to two FILTER clauses (Lines 1 and 2), and the two other edges incident to a_1 correspond to two join clauses (Lines 5 and 7) involving “ $?v_1$ ”. This implies that “ $?v_1$ ” will bind to a resource s that must have a value of the property :fname that matches the keyword “*One-Eyed*” (Line 1) and a value of the property :genre that matches the keyword “*Western*” (Line 2); also, variable “ $?v_4$ ” must bind to a resource o such that there is a triple $(s, \text{hasActor}, o)$ in T (Line 5), and likewise for the other join clause (Line 7).
495

For simplicity, Lines 1-4 adopt a non-standard user-defined predicate *match* that
500 expresses the matches between keywords and literals, and which an implementation will map to a specific technology.

When executed over the RDF graph in Figure 1, such query will return the following triples:

1. $(r_2, \text{:fname}, \text{“One-Eyed Jack”})$
2. $(r_2, \text{:genre}, \text{“Western”})$
3. $(r_3, \text{:aname}, \text{“Marlon Brandon”})$
4. $(r_5, \text{:lname}, \text{“Hollywood”})$
5. $(r_2, \text{:hasActor}, r_3)$
6. $(r_4, \text{:loc}, r_5)$
7. $(r_4, \text{:produces}, r_2)$

505 which is an answer for the keyword-based query K since these triples match all keywords in K and induce a Steiner tree of the RDF graph in Figure 1 that covers all matching nodes.

Finally, if the query returns more than one answer, they are ranked, as briefly discussed in Section 4.4 (see [29][30] for the details). \square

510 The key steps, therefore, are how to select trees to combine by node fusion or by edge addition and how to expand trees by adding new nodes and edges. The previous example illustrates these operations, but it leaves open an important point – how to choose trees, nodes, and edges – which is detailed in Section 4.3.

4.2 Query Graphs

515 This section defines the notion of query graph. The discussion in what follows ignores matches between keywords and class and property labels, which are discussed in Section 4.4.

Let T be an RDF dataset, \mathcal{L} be the set of all literals and $K=\{k_1,\dots,k_n\}$ be a keyword-based query.

520

Definition 1: A query graph for K over T is a node and edge-labeled graph $Q=(N,E,\nu,\lambda)$, where ν labels nodes and λ labels edges, such that:

- N is a set of *match nodes* or *join nodes*.
- E is a set of edges such that all match nodes have in-degree 1 and out-degree 0.
- 525 • λ is such that each edge (a,b) in E is labeled with a property p that occurs in T ; for simplicity, we denote an edge (a,b) in E , labeled with p , as a triple (a,p,b) .
- ν is such that
 - a *match node* in N is labeled with one keyword in K ;
 - a *join node* a in N is labeled with $\nu(a)=\{D_1,\dots,D_m,R_1,\dots,R_n\}$, where:
 - 530 $(a,p_1,b_1)\dots(a,p_m,b_m)$ are all the edges from a ; $(c_1,q_1,a)\dots(c_n,q_n,a)$ are all the edges into a ; D_i is the domain of p_i , for $1\leq i\leq m$; and R_j is the range of q_j , for $1\leq j\leq n$. \square

Example 1, in Section 4.1, illustrated the concept of query graph. Note that the label
 535 of a join node is determined by the domains and ranges of the properties that label the
 edges incident to the node. Also note that Definition 1 assumes that each match node is
 labeled with a single keyword, an assumption adopted just to reduce the complexity of
 the notation and the definitions that follow, and to facilitate understanding the proposed
 algorithm. However, we note that the implementation considers match nodes labeled
 540 with multiple keywords.

Definition 2: A *query tree* (or *forest*) for K over T is a query graph which is a tree (or
 a forest). \square

545 **Definition 3:** Let $Q=(N,E,\nu,\lambda)$ be a query graph for K over T . A SPARQL group graph
 pattern P_Q is induced by Q iff

- for each node a in N , there is a variable $?v_a$; and
- for each edge (a,q,b) of Q , there is a triple pattern in P_Q of the form “ $?v_a$ q $?v_b$ ”,
 if a and b are join nodes, or of the form “ $?v_a$ q $?v_b$ FILTER ($match(?v_b, “M”)$)”,
 550 if b is a match node labeled with “ M ”. \square

As explained in Example 1, for simplicity, the second condition includes a user-
 defined predicate *match* that expresses the matches between keywords and literals. Note
 that, apart from the order of the triple patterns, P_Q is unique. Also note that the same
 555 variable $?v_a$ will be used in the patterns corresponding to the edges $(a,p_1,b_1)\dots(a,p_m,b_m)$
 from a and to the edges $(c_1,q_1,a)\dots(c_n,q_n,a)$ into a .

We leave open the definition of the target clause induced by a query graph Q . It could
 return all, or a subset of the variables bound in the query pattern match (as in the
 SPARQL SELECT query form), or it could return an RDF graph constructed by
 560 substituting variables in a set of triple templates (as in the SPARQL CONSTRUCT query
 form). The first form induces tabular answers, which users preferred in an earlier
 implementation [14]. The following notion of an answer for a query graph Q factors
 out this discussion since it depends only on the group graph pattern P_Q induced by Q .

Definition 4: Let $Q=(N,E,v,\lambda)$ be a query graph for K over T . An *answer* for Q is a
 565 minimal set A_Q of triples in T that satisfies the SPARQL group graph pattern P_Q
 induced by Q . \square

Note that A_Q induces a subgraph of the RDF graph of T and that Q may have more
 than one answer.

570 4.3 A Greedy Algorithm to Translate Keyword-based Queries to SPARQL

Let T be an RDF dataset and $K=\{k_1,\dots,k_n\}$ be a keyword query. This section details
 the proposed algorithm to compile a SPARQL query for K over T whose answers are
 answers for K .

575 Definition 5 formalizes the *node fusion*, *edge addition*, and *tree expansion*
 operations. To avoid an awkward notation, in what follows, tree expansion is broken
 into two operations that depend on the direction of the edge added.

Definition 5: Let $Q=(N,E,v,\lambda)$ be a query graph for K over T . Let a_i be a node labeled
 580 with $A_i = \{A_{i,1}, \dots, A_{i,n_i}\}$, for $i=1,2$, and p be a property with domain D_p and range
 R_p . Assume that a_1 and a_2 belong to different trees t_1 and t_2 .

- a) The *fusion* of nodes a_1 and a_2 replaces a_1 and a_2 by a new join node c , labeled
 with $\{A_{1,1}, \dots, A_{1,n_1}, A_{2,1}, \dots, A_{2,n_2}\}$.
- b) The *addition of a join edge*, labeled with p , from a_1 to a_2 relabels a_1 with
 585 $\{A_{1,1}, \dots, A_{1,n_1}, D_p\}$ and a_2 with $\{A_{2,1}, \dots, A_{2,n_2}, R_p\}$, and adds the join edge
 (a_1,p,a_2) to the query graph.
- c) The *expansion* of tree t_1 by the addition of a join edge, labeled with p , from a_1
 relabels a_1 with $\{A_{1,1}, \dots, A_{1,n_1}, D_p\}$, adds a new node c , labeled with $\{R_p\}$, and
 adds the join edge (a_1,p,c) to the query graph.
- 590 d) The *expansion* of tree t_2 by the addition of a join edge, labeled with p , into a_2
 relabels a_2 with $\{A_{2,1}, \dots, A_{2,n_2}, R_p\}$, adds a new node c , labeled with $\{D_p\}$, and
 adds the join edge (c,p,a_2) to the query graph. \square

595 Algorithm 1 summarizes the basic steps of the strategy, illustrated in Section 4.1. In the algorithm, a *property match* for K over T is a pair (p, k_i) such that there is at least one triple in T of the form (s, p, o) such that o matches $k_i \in K$.

Algorithm 1: TRANSLATEKEYWORDQUERY

Input: T – an RDF dataset

K – a keyword-based query over T

Output: Φ – a query for K over T that returns answers for K

Step 1: Match the keywords in K with literals in T , creating a set \mathcal{S} of property matches (p, k_i) such that p is a property that occurs in T , and $k_i \in K$.

Step 2: Use the set \mathcal{S} of matches found in Step 1 to construct an *initial query forest* as follows: for each match (p, k_i) in \mathcal{S} , where D is the domain of p , the forest has a join node a , labeled with $\{D\}$, a match node b , labeled with “ k_i ”, and an edge (a, p, b) .

Step 3: Reduce the number of trees of the query forest by using *node fusion*, *edge addition*, and *tree expansion*.

Step 4: Select the tree with the largest number of keyword matches, construct a SPARQL query Φ from the selected tree, whose WHERE clause is as in Def. 3, and output Φ .

Step 1 is described in Section 4.4. Step 2 is simple and was already illustrated in Section 4.1. In what follows, we cover in detail Step 3, the core of Algorithm 1, and conclude with Step 4.

600 Step 3 selects operations based on scores defined as follows.

Definition 6: Let $Q=(N, E, \nu, \lambda)$ be a query graph for K over T . Let a_i be a node labeled with $\{A_{i,1}, \dots, A_{i,n_i}\}$, for $i=1,2$, and p be a property with domain D_p and range R_p .

a) *Node Fusion Score*: assesses when to combine a_1 and a_2 into a single node:

605
$$\text{node_fusion_score}(a_1, a_2) = J(A_{1,1}, \dots, A_{1,n_1}, A_{2,1}, \dots, A_{2,n_2})$$

b) *Edge Addition Score*: assesses when to add a join edge, labeled with p , outgoing from a_1 and incoming into a_2 :

$$\begin{aligned} \text{edge_addition_score}(a_1, p, a_2) \\ = C(A_{1,1} \cap \dots \cap A_{1,n_1}, D_p) \times C(A_{2,1} \cap \dots \cap A_{2,n_2}, R_p) \end{aligned}$$

610 c) *Outgoing Tree Expansion Score*: assesses when to add a join edge, labeled with p , outgoing from a_1 :

$$\text{outgoing_tree_expansion_score}(a_1, p) = C(A_{1,1} \cap \dots \cap A_{1,n_1}, D_p)$$

d) *Incoming Tree Expansion Score*: assesses when to add a join edge, labeled with p , incoming into a_2 :

$$615 \quad \text{incoming_tree_expansion_score}(a_2, p) = C(A_{2,1} \cap \dots \cap A_{2,n_2}, R_p)$$

.□

All these scores can then be estimated using KMV-synopses for such sets – and this was the reason for adopting KMV-synopses.

In more detail, the node fusion operation uses the node fusion score to decide when
620 to combine two nodes, a_1 and a_2 , based on how similar all sets that label the nodes are; note that the node fusion score depends on the n-way Jaccard similarity measure, since all sets should be considered equally relevant.

The edge addition operation uses the edge addition score to decide when to add an edge between two nodes, a_1 and a_2 , labelled with a property p , based on how the
625 intersection $A_{1,1} \cap \dots \cap A_{1,n_1}$ of all sets that label a_1 are similar to the domain of p and, simultaneously, how the intersection $A_{2,1} \cap \dots \cap A_{2,n_2}$ of all sets that label a_2 are similar to the range of p ; edge addition depends on set containment, since edge addition is similar to a query that locates sets similar to $A_{1,1} \cap \dots \cap A_{1,n_1}$, and likewise for $A_{2,1} \cap \dots \cap A_{2,n_2}$.

630 Finally, the tree expansion operation uses the outgoing/incoming tree expansion scores much in the same way that the edge addition operation uses the edge addition score; the difference lies in that the added edge, labelled with property p , is such that the domain of p is similar to the intersection $A_{1,1} \cap \dots \cap A_{1,n_1}$ of all sets that label a_1 , in the case of the outgoing tree expansion score, and likewise the range of p is similar
635 to the intersection $A_{2,1} \cap \dots \cap A_{2,n_2}$ of all sets that label a_2 , in the case of the outgoing tree expansion score.

More precisely, the following proposition lists properties of such scores and might be skipped on a first reading ($Pr[S]$ denotes the probability of S),

640 **Proposition 1:** Let T be an RDF dataset. Let $A_{i,1}, \dots, A_{i,n_i}$, for $i=1,2$, and B_1, B_2 be sets of IRIs, and p be a property with domain D_p and range R_p .

- a) Randomly draw an element s from $A_{1,1} \cup \dots \cup A_{1,n_1} \cup A_{2,1} \cup \dots \cup A_{2,n_2}$. Then, the probability that the element s is in $A_{1,1} \cap \dots \cap A_{1,n_1} \cap A_{2,1} \cap \dots \cap A_{2,n_2}$ is given by $J(A_{1,1}, \dots, A_{1,n_1}, A_{2,1}, \dots, A_{2,n_2})$.
- 645 b) Randomly draw s from $B_1 \cup D_p$. Then, the probability that s is in $B_1 \cap D_p$, knowing that s is in B_1 , is given by $C(B_1, D_p)$.
- c) Randomly draw o from $B_2 \cup R_p$. Then, the probability that o is in $B_2 \cap R_p$, knowing that o is in B_2 , is given by $C(B_2, R_p)$.
- d) Randomly draw s from $B_1 \cup D_p$ and o from $B_2 \cup R_p$. Then, the probability that
650 s is in $B_1 \cap D_p$ and o is in $B_2 \cap R_p$, knowing that s is in B_1 and o is in B_2 , is given by $C(B_1, D_p) \times C(B_2, R_p)$.

Proof

(a) Randomly draw s from $A_{1,1} \cup \dots \cup A_{1,n_1} \cup A_{2,1} \cup \dots \cup A_{2,n_2}$. Then, by definition of the Jaccard similarity, we have

$$Pr \left[s \in \bigcap_{i=1,2}^{j=1,\dots,n_1} A_{i,j} \right] = \frac{|\bigcap_{i=1,2}^{j=1,\dots,n_1} A_{i,j}|}{|\bigcup_{i=1,2}^{j=1,\dots,n_1} A_{i,j}|} = J(A_{1,1}, \dots, A_{1,n_1}, A_{2,1}, \dots, A_{2,n_2})$$

655 (b) Randomly draw s from $B_1 \cup D_p$. The probability σ we want to compute is:

$$\sigma = Pr[s \in B_1 \cap D_p \mid s \in B_1]$$

By definition of conditional probability, we have

$$\sigma = \frac{Pr[s \in B_1 \cap D_p]}{Pr[s \in B_1]}$$

Since s is drawn from $B_1 \cup D_p$, we have

$$Pr[s \in B_1 \cap D_p] = \frac{|B_1 \cap D_p|}{|B_1 \cup D_p|}$$

$$Pr[s \in B_1] = \frac{|B_1|}{|B_1 \cup D_p|}$$

From the above equalities and by definition of set containment, we have

$$\sigma = \frac{|B_1 \cap D_p|}{|B_1 \cup D_p|} \times \frac{|B_1 \cup D_p|}{|B_1|} = \frac{|B_1 \cap D_p|}{|B_1|} = C(B_1, D_p)$$

(c) (Follows likewise).

660 (d) The probability ρ we want to compute is

$$\rho = Pr[s \in B_1 \cap D_p, o \in B_2 \cap R_p \mid s \in B_1, o \in B_2]$$

By definition of conditional probability, we have

$$\rho = \frac{Pr[s \in B_1 \cap D_p, o \in B_2 \cap R_p]}{Pr[s \in B_1, o \in B_2]}$$

By the independence of the drawings, we have

$$\rho = \frac{Pr[s \in B_1 \cap D_p] \times Pr[o \in B_2 \cap R_p]}{Pr[s \in B_1] \times Pr[o \in B_2]}$$

Then, as in (b), we immediately have that

$$\rho = C(B_1, D_p) \times C(B_2, R_p)$$

which concludes the proof. \square

665

Returning to Step 3 of Algorithm 1, its implementation is limited by the following result. Define the *MQF Problem* as: “Given a query forest $Q=(N,E,\nu,\lambda)$ for K over T , and minimal bounds for the scores, find a minimal forest obtained by repeatedly applying the *node fusion*, *edge addition* and *tree expansion* operations to Q , provided

Algorithm 2: REDUCEQUERY

Input: T – an RDF dataset
 p_1, \dots, p_n – the list of properties that occur in T
 Q – an initial query forest
 δ – a minimum threshold
 η – the max number of reduction cycles allowed
Output: Q – a modified query forest, possibly with fewer trees

```

1 begin
2   count = 0;
3   while  $Q$  has more than 1 tree and count  $\leq \eta$  do
4     begin
5       COMBINETREESBYNODEFUSION( $T, Q, \delta; Q$ );
6       If  $Q$  has a single tree then return  $Q$ ;
7       COMBINETREESBYEDGEADDITION( $T, (p_1, \dots, p_n), Q, \delta; Q$ );
8       if  $Q$  has a single tree then return  $Q$ ;
9       EXPANDTREE( $T, (p_1, \dots, p_n), Q, \delta; Q$ );
10      count = count + 1;
11    end
12    CLEANQUERY( $Q$ );
13  return  $Q$ ;
14 end
```

670 that the scores of these operations are above minimal bounds”. We note that the use of
minimal bounds in this definition is just a theoretical device to allow formulating the
MQF Problem as a decision problem and not as an optimization problem.

Proposition 2: The MQF Problem is NP-complete.

675 (*Proof Sketch:* By a reduction from the minimal Steiner tree problem, as in [14]).

In the face of Proposition 2, the rest of this section introduces a heuristic to
implement Step 3, based on the scores introduced in Definition 6, estimated using
KMV-synopses.

680 Algorithm 2 – REDUCEQUERY implements Step 3 of Algorithm 1 by combining
distinct trees by node fusion (Line 5) or edge addition (Line 7), and by tree expansion
(Line 9). Note that a tree expansion operation (Line 9) may add a new edge and a new
node that may end up not being used in later cycles to combine trees. This can be
detected, when the loop (in Lines 3-11) finishes, by checking if there is a node, with
685 only one incident edge, which is not a matching node. A cleaning operation (Line 12)
will then eliminate such nodes and their incident edges. Finally, the number of cycles
is limited to η to avoid adding too many new edges, which might lead to less meaningful
queries. The constant η was empirically determined during the experiments described
in Section 6. Indirectly, η places an upper bound on the length of the paths between two
690 nodes in the query forest. This is justified since, as argued in [33], long paths may
express unusual relationships, which might be misinterpreted by users.

We now describe the four procedures used in Algorithm 2 – REDUCEQUERY.

Algorithm 3 – COMBINE TREES BY NODE FUSION implements the node fusion operation
and uses the node fusion score to decide when to combine two nodes. Algorithm 3
695 selects two nodes to apply node fusion in decreasing order of node fusion scores (Lines
4 and 13). Lines 11-13 are necessary to accommodate the new node obtained by node
fusion. Finally, the minimum score for the node fusion score (Lines 3 and 12) tries to
reduce the number of pairs added to the list L and, consequently, the number of cycles
of Algorithm 3.

700

Algorithm 3: COMBINE TREES BY NODE FUSION

Input: T – an RDF dataset
 Q – a query forest
 δ – a minimum threshold
Output: Q – a reduced query forest

```
1 begin
2   create a list  $L$  of pairs of join nodes, each from a different tree,
3     with node fusion scores above the threshold  $\delta$ ;
4   order  $L$  in decreasing order of node fusion score;
5   while  $L$  is not empty do
6     begin
7       apply node fusion to  $(a_1, a_2)$ , creating a new node  $a_3$ ,
8         and modifying the forest  $Q$  accordingly;
9       remove from  $L$  any pair involving  $a_1$  and  $a_2$ 
10      and any pair of nodes that are now in the same tree;
11      add to  $L$  all new pairs involving the new node  $a_3$ ,
12        with node fusion scores above the threshold  $\delta$ ;
13      reorder  $L$  in decreasing order of node fusion score;
14    end;
15  return  $Q$ ;
16 end
```

Algorithm 4 – COMBINE TREES BY EDGE ADDITION implements the edge fusion operation and uses the edge addition score to decide when to add an edge between two nodes. Line 5 avoids considering the combination of two trees that have a score which is too low, which would possibly lead to a query with too few answers, or no answer at all. Line 13 is necessary because, later on, the new tree $C_{1,2}$ might in turn be combined with other trees.

Algorithm 5 – EXPAND TREE implements the tree expansion operation and uses the outgoing/incoming tree expansion scores to decide when to add an outgoing/incoming edge to a node.

Algorithm 6 – CLEAN QUERY receives a query forest and eliminates unnecessary edges so that all terminal nodes of the modified query forest are match nodes.

715

Algorithm 4: COMBINETREESBYEDGEADDITION

Input: T – an RDF dataset
 p_1, \dots, p_n – the list of properties that occur in T
 Q – a query forest
 δ – a minimum threshold

Output: Q – a reduced query forest

```
1 begin
2   mark all trees of  $Q$  as unprocessed;
3   while  $Q$  has more than one tree and
4     there is a pair  $(C_1, C_2)$  of unprocessed trees of  $Q$ 
5     such that  $tree\_edge\_combination\_score(C_1, C_2) \geq \delta$  do
6     begin
7       select the pair  $(C_1, C_2)$  of unprocessed trees of  $Q$ 
8         with the highest  $tree\_edge\_combination\_score(C_1, C_2)$ ;
9       select  $a_i \in C_1, a_k \in C_2$ , for  $1 \leq i \neq k \leq 2$ , and a property  $p_j$  such that  $(a_i, p_j, a_k)$ 
10        has the highest  $edge\_combination\_score(a_i, p_j, a_k)$ ;
11        let  $a_i$  and  $a_k$  be labelled with  $A_i$  and  $A_k$ , respectively;
12        add  $(a_i, p_j, a_k)$  to  $Q$ , combining  $C_1$  and  $C_2$  into a single tree  $C_{1,2}$ ;
13        relabel  $a_i$  with  $A_i \cup \{D_j\}$  and  $a_k$  with  $A_k \cup \{R_j\}$ ,
14        where  $D_j$  and  $R_j$  are the domain and range of  $p_j$ , respectively;
15        mark  $C_{1,2}$  as unprocessed;
16    end;
17    return  $Q$ ;
18 end
```

By induction on the number of node fusions, edge additions and tree expansions applied, and by Definitions 3 and 4, we can prove the correctness of Algorithm 1.

720

Proposition 3: Let $Q=(N,E,v,\lambda)$ be the tree selected in Step 4 of Algorithm 1. Let A be an answer for Q over T . Then, A is an answer for K over T . \square

As for the overall complexity, Algorithm 2 executes at most η cycles. In each cycle, node fusions are tried, then edge additions and, if the forest has not been reduced to a single tree, a tree expansion. Let $|K|$ be the number of keywords of the query. Let N_p be the number of properties that occur in the RDF graph (i.e., the number of IRIs that denote properties). In the worst case, one tree expansion is executed per cycle, which adds a new join node. The initial number of join nodes is at most $|K|$. Hence, in the i^{th} cycle (starting with $i=0$), there are at most $(|K|+i)$ join nodes. Then, there are at most $(|K|+i)^2$ possible node fusions, $(|K|+i)^2 \times 2N_p$ possible edge additions (we have to multiply by 2 since we also have to try the inverse of each property), and $(|K|+i) \times 2N_p$

725
730

possible tree expansions. Hence, since η is a constant, in the worst case, the time complexity of Algorithms 2 and 3 is $O(|K|^2 \times N_P)$.

735 This concludes the discussion of Step 3 of Algorithm 1.

Algorithm 5: EXPANDTREE

Input: T – an RDF dataset
 p_1, \dots, p_n – the list of properties that occur in T
 Q – a query forest
 δ – a minimum threshold
Output: Q – a reduced query forest

```

1 begin
2   mark all trees of  $Q$  as unprocessed;
3   while  $Q$  has more than one tree and
4     there is an unprocessed tree  $C$  of  $Q$  do
5     begin
6       select the join node  $a$  in  $C$  and the predicate  $p_j$  in  $T$ 
7         with the highest of the scores:
8            $out\_edge\_score(a, p_j)$  or  $in\_edge\_score(a, p_j)$ ;
9       add a new node  $b$  to  $C$ ;
10      add the join edge  $(a, p_j, b)$  (or  $(b, p_j, a)$ ) to  $C$ ;
11      let  $a$  be labelled with  $A$ ;
12      let  $D_j$  and  $R_j$  be domain and range of  $p_j$ , respectively;
13      relabel  $a$  with  $A \cup \{D_j\}$  and  $b$  with  $\{R_j\}$ 
14        (or  $a$  with  $A \cup \{R_j\}$  and  $b$  with  $\{D_j\}$ );
15      mark  $C$  as processed;
16    end
17 end
18
```

Algorithm 6: CLEANQUERY

Input: Q – a query forest, possibly with join nodes as terminals
Output: Q – a modified query forest whose terminals are match nodes

```

1 begin
2   while there is a terminal node  $a$  which is not a match node do
3     begin
4       /* since  $Q$  is a forest and  $a$  is terminal,
5         there is just one edge incident to  $a$  */
6       delete the edge incident to  $a$ ;
7       delete the join node  $a$ ;
8     end
9 end
```

We conclude this section by returning to Step 4 of Algorithm 1. It suffices to recall that this step selects the tree T with the largest number of keyword matches and constructs the final SPARQL query Φ from T . Definition 3 explained how to construct the WHERE clause of Φ from T , as already illustrated towards the end of Example 1 of Section 3. The discussion after Definition 3 indicated how to synthesize the target clause of Φ . In the SPARQL SELECT query format, Φ could return all, or a subset of the variables bound in the WHERE clause of Φ .

Algorithm 7 – COMPILERQUERY summarizes these observations, for the SELECT query format with all variables bound in the WHERE clause.

Algorithm 7: COMPILERQUERY

Input: $Q=(N,E,v,\lambda)$ – a query forest

Output: Φ – the final SPARQL query

```

1  begin
2    select the tree  $T=(N,E,v,\lambda)$  in  $Q$  with
3      the largest number of keyword matches;
4    construct the WHERE clause of  $\Phi$  as follows:
5      for each node  $a$  in  $N$ , create a variable  $?v_a$ ;
6    for each edge  $(a,q,b)$  of  $T$ ,
7      create a triple pattern in  $P_Q$  of the form “ $?v_a q ?v_b$ ”,
8      if  $a$  and  $b$  are join nodes,
9      or of the form “ $?v_a q ?v_b \text{ FILTER } ( \text{match}(?v_b, “M”) )$ ”,
10     if  $b$  is a match node labeled with “ $M$ ”.
11   construct the target list of  $\Phi$  with all variables used in WHERE clause;
12 end
```

4.4 Additional Remarks on the Translation of Keyword-based Queries to SPARQL

Treatment of Class and Property Labels. First, observe that a keyword may match the label of a class or property, which alters the interpretation of the keyword-based query. For example, if *Actor* is declared as a class with label “*actor*”, then the keyword-based query $K = \{actor, Washington\}$ is interpreted as requesting instances of the class *Actor* that have properties that match “*Washington*”.

Assume that a keyword matches the label of class c , declared in the RDF dataset T (labels are identified when T is scanned in the KMV-synopses computation). Briefly, this is captured by modifying the definition of query graph to accommodate classes in the query graphs and the SPARQL group graph pattern induced by a query graph. The experience with previous implementations of keyword search tools [14, 23, 30] showed that class and predicate matches in fact forces the query translation process to focus on instances of the matched classes or properties, which reduces the overall query elapsed time.

The modifications to account for keywords that match property labels are entirely similar. The use of other terms of the RDF Schema vocabulary is outside the scope of this article and is discussed elsewhere.

Use of Ranking. We now briefly discuss two questions: (1) how to define ranking measures specifically for RDF graphs; (2) how to use these measures to help compute and rank answers of keyword queries over RDF graphs.

To address the first problem, we proposed a family of importance measures for RDF graphs in [29][30], collectively called *InfoRank*, that combines three intuitions: (I) “important things have lots of information about them”; (II) “important things are surrounded by other important things”; (III) “few important relations (e.g. friends) are better than many unimportant relations (e.g. acquaintances)”. *InfoRank* requires neither the manual assignment of weights to object properties nor a training dataset to use as input to a learning algorithm.

Let T be a set of RDF triples. Recall from Section 3.1 that it is possible to identify the set C of classes observed in T , the set P of object properties observed in T , the set L of literals observed in T , and the set R of (class) instances observed in T .

The *informativeness* of an instance $r \in R$, denoted $IW(r)$, is defined as the number of triples of the form $(r, p, v) \in T$, where $v \in L$, that is, the number of property values that describe instance r . Based on instance informativeness, we say that “important classes usually have informative instances” and “important properties are usually those connecting informative instances”. More precisely, the *InfoRank* of a class $c \in C$, denoted $IR(c)$, is defined as the maximum value of $IW(r)$ of all instances of class c .

785 Likewise, the *InfoRank* of an object property $p \in P$, denoted $IR(p)$, is defined as the maximum value of $IW(r)+IW(s)$ of all triples of the form $(r,p,s) \in T$.

Note that we used only *Intuition I* to rank classes and object properties. However, we propose a combination of the three intuitions to rank class instances.

Let $r,s \in R$ and $p \in P$. Assume that $(r,p,s) \in T$ or $(s,p,r) \in T$, that is, ignore the direction
790 of the object property p . The *normalized weight* of (r,p) , denoted $W(r,p)$, is defined as:

$$W(r,p) = IR(p) / \sum_{q \in P \text{ and } ((r,q,t) \in T \text{ or } (t,q,r) \in T)} IR(q) \quad (26)$$

Then, the weighted *PageRank* score of an instance r , denoted $PR_W(r,i)$, is recursively defined as:

$$PR_W(r,0) = 1/N \quad (27)$$

$$PR_W(r,i) = \frac{1-\alpha}{N} + \alpha \sum_{(r,p,s) \in T \text{ or } (s,p,r) \in T} PR_W(s,i-1) * W(r,p) \quad (28)$$

where N is the total number of nodes in T and α is a dumping factor (usually set to 0.85).

795 The *InfoRank* score of an instance r , denoted $IR(r)$, is the *PageRank* score of r after a fixed number x of iterations, $PR_W(r,x)$, weighted by the informativeness of r , $IW(r)$:

$$IR(r) = PR_W(r,x) * IW(r) \quad (29)$$

We conclude with a brief discussion about how to use *InfoRank* in the context of the process described in Section 4.3 to compute answers to RDF keyword queries. Recall that Step 1 of Algorithm 1 matches keywords with property values, and also with class
800 (or object property) labels or descriptions.

First, class and property labels or descriptions matches have priority over property value matches. Whenever a keyword matches more than one class (or object property) label or description, Step 1 of Algorithm 1 ranks all such classes (and object properties) by descending order of a linear combination of the match score values with the pre-computed *InfoRank* score values for the classes (or object properties) and considers
805 only the topmost class (or object property). Likewise, Step 1 of Algorithm 1 ranks the property value matches in decreasing order of their combined scores and considers only the topmost match.

Finally, Step 4 of Algorithm 1 ranks the answers of a query using again a linear
810 combination of match score values with pre-computed *InfoRank* score values. This is
implemented by modifying the final SPARQL query to also retrieve the pre-computed
InfoRank scores for the instances observed in an answer and to include an *order by*
clause that ranks the answers accordingly.

815 **Beyond synopses and ranking.** There are special cases of keyword queries where
synopses and ranking measures do not need to be used in the keyword query translation
process into the SPARQL query. The first one consists of queries with a single keyword
that matches a class label. The second case includes queries with two keywords, where
820 a keyword matches the label of a resource r , such that the triple $(r, \text{rdf:type}, c)$ is in T ,
and the other keyword matches the label of an object property p , where the domain and
range of p is c . Note that the triple $(r, \text{rdf:type}, c)$ is in T implies that c is an observed
class in T . We now use two examples to illustrate these cases.

Let us consider that the keyword-based query system is running on top of DBpedia.
Suppose that the system receives the keyword query $K = \{\textit{World Heritage Site}\}$ as
825 input. Hence, the matching process finds that this keyword matches the label of class
`dbo:WorldHeritageSite`, since the triple $(\text{dbo:WorldHeritageSite}, \text{rdfs:label}, \text{"World Heritage Site"})$ is in DBpedia. Next, the system directly synthesizes a SPARQL query whose
WHERE clause corresponds to the triple pattern $(?r, \text{rdf:type}, \text{dbo:WorldHeritageSite})$,
where the variable $?r$ binds the class resources.

830 Suppose now that the system receives the keyword query $K = \{\textit{goofy}, \textit{creator}\}$. So,
the matching process finds that the keyword *goofy* matches the label of the resource
`dbr:Goofy`[†], and the keyword *creator* matches the label of the object property `dbo:creator`[‡].
Next, the system detects that `dbr:Goofy` is a resource of class `dbo:Person`, and `dbo:creator`
has resources of `dbo:Person` both in the domain and range. Thus, the system is unable to
835 resolve this ambiguity, i.e., it cannot decide if `dbr:Goofy` belongs to the domain or to the
range of `dbo:creator`. To deal with this issue, the system compiles a WHERE clause with
the following triple pattern

[†] <http://dbpedia.org/page/Goofy>

[‡] <http://dbpedia.org/ontology/creator>

```
840      { ?r rdfs:label "Goofy"@en .  
          ?x dbo:creator ?y  
          FILTER (sameTerm(?r, ?x) || sameTerm (?r, ?y) }
```

Finally, the system runs the compiled SPARQL query and ranks the answers.

5. Comparison with a Schema-based RDF Keyword Search Tool

845 This section describes a set of experiments that compares the schema-less approach proposed in this article with a state-of-the-art schema-based RDF keyword search tool, adopted as baseline. The experiments are based on a RDF keyword search benchmark [32], with: two RDF datasets triplified from the IMDb[§] and Mondial^{**} databases; the RDF schema of each of these RDF datasets; two lists of keyword-based queries, one for each of these RDF datasets, and their relevant answers. The benchmark is openly available at GitHub [31].

850 5.1 Benchmark

The benchmark adopted in this part of the experiments was inspired by Coffman's benchmark [10], created to evaluate keyword search tools over relational databases. Coffman's benchmark is based on data and the relational schemes for IMDb, Mondial, and Wikipedia; each dataset has 50 keyword-based queries and their expected answers.

855 However, the expected answers in Coffman's benchmark do not always cover all possibilities and are somewhat arbitrary. For example, the keyword-based query {*niger*} over Mondial had as the expected answer only the instance of the class *River* labeled as "*Niger*". However, the instances of the classes *Country* and *Province* labeled as "*Niger*" should also be considered valid answers. In fact, the user can refine the query
860 as {*River, niger*}, if s/he is indeed interested in the Niger River. Furthermore,

[§] <http://www.imdb.com>

^{**} <http://www.dbis.informatik.unigoettingen.de/Mondial>

Coffman's benchmark group queries by topics, so that all queries within the same topic are quite similar and redundant with respect to testing the capability of the keyword search tools.

865 The adopted benchmark differs from Coffman's in three aspects: (1) it uses triplified versions of Mondial and IMDb; (2) it includes only keyword-based queries that have answers that explore the structure of the RDF graph; (3) for each keyword-based query, it contains a ranked list of answers, created with the help of the graph-based algorithm described in [32].

870 In more detail, Table 1 summarizes the characteristics of the triplified versions of the Mondial and IMDb databases included in the benchmark. We note that the Mondial RDF graph is much more complex than that of IMDb. However, the size of IMDb is significantly larger than the size of Mondial, in terms of the number of triples. For both datasets, we included the *InfoRank* property values computed in [29]. The schema and data of Mondial dataset are available at <<https://www.dbis.informatik.uni-goettingen.de/Mondial>>. Regarding IMDb, the RDF Schema and data are available at the QUIRA Official Page^{††}.
875

^{††} <https://sites.google.com/view/quira/>

Table 1. Statistics – Mondial and IMDb Datasets.

Characteristics	Mondial	IMDb
N-Triples File Size	27.6 MB	18.1 GB
Triple Types		
Class instances	37,468	32,349,586
Class instances label declarations	7,620	28,962,103
rdfs:Class declarations	-	25
Classes	27	25
subClassOf axioms	-	16
Object property	32	35
Datatype property	27	89
Metadata labels declarations	86	147
Ranking datatype properties	70,611	25,968,919
Distinct indexed property values	45,325	10,571,370
Total number of triples	266,985	201,622,903
Jena properties		
Database size	0.99 GB	42 GB
Lucene index size	4.11 MB	4.1 GB
Saving data time (upload + index)	16.8 sec	~ 4 h

As in [10], the benchmark keyword-based queries are not real user queries extracted from a search engine logs, yet they reflect distinct information needs. The benchmark has 24 keyword-based queries for Mondial and 40 for IMDb, grouped according to the expected graph patterns in their answers, as shown in Tables 4 and 5. The average number of terms per keyword-based query is 3.42 for Mondial and 4.38 for IMDb.

By construction, all keyword-based queries have non-empty answers. For each keyword-based query K , the benchmark has a ranked list $S_{K,1}, \dots, S_{K,m}$ of sets of triples of the underlying dataset, called the *solution generators* for K . Each solution generator $S_{K,p}$ has a set of literal nodes, called *seeds*, that match the keywords in K . The solution generators computed for each keyword query are available at <https://figshare.com/s/ef9aed9657255a01c008> in the path `src/main/resources/benchmarks/ER2020`.

To test an RDF keyword search algorithm \mathcal{A} , one would use solution generators as follows. For each keyword-based query K of the benchmark, one would submit K to \mathcal{A} and obtain one or more answers $A_{K,1}, \dots, A_{K,n}$. Each answer $A_{K,q}$ should be considered *relevant* iff $A_{K,q}$ induces a connected subgraph of one of the solution generators $S_{K,p}$ of K and $A_{K,q}$ includes all seeds of $S_{K,p}$ as nodes. One might assign a score to $A_{K,q}$ based on the number of keywords of K that $A_{K,q}$ matches (i.e., the number of seeds of $S_{K,p}$), the

895 number of triples of $A_{K,q}$ and the position of $S_{K,p}$ in the ranked list of solution generators
for K . The exact score function is user-defined and outside the scope of the benchmark.

5.2 Experimental Setup

KMV-synopses RDF keyword search tool. We implemented an RDF keyword search
tool based on the proposed algorithm, which we refer to as the *KMV-synopses RDF*
900 *keyword search tool*, or simply the *KMV-synopses tool*, to differentiate it from the
baseline tool described below. We used Java 14 to implement the tool and Lucene^{‡‡},
which is hosted with Jena, to index the datatype property values, including `rdfs:label`
values. This feature permitted combining SPARQL queries and full-text search.

The tool precomputes the KMV-synopses and stores them together with the dataset.
905 When the dataset is opened, the tool loads the KMV-synopses into main memory to
speed up the compilation of keyword-based queries. If necessary, the tool recomputes
the KMV-synopses from time-to-time, much in the same way that database systems
recompute statistics, as already pointed out in Section 2.

To compute KMV-synopses, we adopted as hash function $h(x) = \text{MD5}(x) \% M^p$, where
910 M is the number of class instances in the dataset. $\text{MD5}(x)$ is computed by calling the
static method `MessageDigest.getInstance("MD5")`^{§§}. We used $k=16,384$ for IMDb, and
 $k=8,192$, for Mondial.

Table 3 shows the time and space the KMV-synopses tool required to compute the
KMV-synopses for the Mondial and IMDb datasets, which are consistent with the fact
915 that IMDb is 3 orders of magnitude larger than Mondial. Note that, even for IMDb, it
would indeed be feasible to recompute the KMV-synopses from time-to-time, if IMDb
is updated.

^{‡‡} <http://lucene.apache.org/>

^{§§} <https://docs.oracle.com/javase/7/docs/api/java/security/MessageDigest.html>

920

Table 3 - Space and time required to construct and store KMV-synopses.

Dataset	Time	Space
Mondial	9 sec	439.0 KB
IMDb	152 min	30.6 MB

For each keyword-based query, the KMV-synopses tool returns a ranked list of answers.

925

Baseline. As baseline, we adopted the schema-based RDF keyword search tool described in [14], which we had full control and could test the performance in a carefully controlled environment. This baseline tool is fast and had good precision in earlier experiments [14][23]. For simplicity, we refer to it as the *baseline tool*. We extended the original implementation, which used Oracle RDF, to Jena to be able to run the benchmark queries and have a fair comparison baseline.

930

For each keyword-based query, the baseline tool returns an ordered list of answers, but it does not adopt any particular ranking strategy.

935

Metrics. To measure the effectiveness of the tools, we used the same metrics considered in [10]: *Mean Average Precision (MAP)*, *Top-1*, and *Mean Reciprocal Rank (MRR)*. The *average precision* for a query is the average of the precision values computed after each relevant answer is retrieved (and assigning a precision of 0.0 to any relevant answer not retrieved). If the average precision value of a query is 0, then we consider it a *failed query*. *MAP* is the average of precision across all queries and provides a single-figure measure of quality across recall levels. It has been shown that, among the IR measures, *MAP* has especially good discrimination and stability. The number of *top-1* relevant answers is the number of queries for which the first answer belongs to the highest-ranked relevant answer retrieved by the system. The *reciprocal rank* is the reciprocal of the highest-ranked relevant answers for a given query. *MRR* is a statistical measure for evaluating any system that produces a ranked list of answers for every query. It considers the highest-ranked relevant answer retrieved for each query. The *Reciprocal Rank (RR)* of a query is calculated by reversing the rank of the highest-ranked relevant answer retrieved by the system. *MRR* is computed by averaging

940

945

RR values over all queries. *Top-1* and *MRR* metrics are known to be poorly stable, but
950 they indicate the quality of the top-ranked answers.

Let K be a keyword-based query, with a ranked list $S_{K,1}, \dots, S_{K,m}$ of solution
generators, defined in the benchmark. Recall that each of the tools returns an ordered
list $\mathbf{A}=(A_{K,1}, \dots, A_{K,n})$ of answers for K . Then, we tested each answer $A_{K,q}$ in the ordered
list for relevancy, as explained at the end of Section 5.3, finding the $S_{K,p}$ (if it exists) for
955 which $A_{K,q}$ is relevant, and applied the metrics described earlier. In particular, we used
the ranked positions q and p to compute the *MRR* metric, and we only tested if the first
answer $A_{K,1}$ is relevant with respect to the first solution generator $S_{K,1}$ to compute the
top-1 metric.

We computed the Top-1 and the MRR metrics only for the proposed tool since the
960 baseline tool returns an unordered list of answers. For this reason, we also did not use
the normalized discounted cumulative gain (NDCG) in these experiments. Indeed, it
would be unfair to compare the approach proposed in this article with the baseline tool,
using ranking metrics such as Top-1, MRR, and NDCG.

965 **Hardware and software setup.** All tests were executed on a desktop machine with OS
Windows 10 Pro, a quad-core processor Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz,
16GB of RAM. To store and manage the RDF dataset, we used the component TDB2
of Apache Jena for RDF (<https://jena.apache.org>). Apache Jena Fuseki (a SPARQL
server) ran on a server machine with OS GNU/Linux Ubuntu 16.04.6 LTS, a quad-core
970 processor Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz, 64 GB of RAM and SSD
1TB. The critical drawbacks of executing queries in Jena are the size of the heap
memory and the timeout. For our tests, we configured the value of heap memory as
JVM_ARGS: --Xmx60G and set the query timeout to 2 hours.

5.3 Experimental Evaluation

975 We examined the performance of the KMV-synopses tool and compare it with the
baseline described in Section 5.2.

Table 4 - Experiments with Mondial.

Benchmark		Failed Queries		AP		Top-1	RR
Query Groups	#Queries	BL	KMV	BL	KMV	KMV	KMV
A- Retrieve resources using metadata and value matches	4	-	-	1.00	1.00	1.00	1.00
B- Join of instances of different classes using metadata and value matches	4	-	-	1.00	1.00	1.00	1.00
C- Join resources of the same class using value matches	4	4	-	0.00	1.00	1.00	0.75
D- Join two same class resources to resources of another class	4	4	-	0.00	1.00	1.00	1.00
E- Join a pair of resources from different classes to elements of another class through intermediary nodes	4	-	-	1.00	1.00	1.00	0.55
F- Join resources of various classes	4	2	1	0.50	0.75	0.75	0.43
OVERALL	24	10	1	0.58	0.96	0.96	0.79

BL –baseline tool KMV – KMV-synopses tool

AP- Average Precision

RR- Reciprocal Rank

Table 5 - Experiments with IMDB.

Benchmark		Failed Queries		AP		Top-1	RR
Query Groups	#Queries	BL	KMV	BL	KMV	KMV	KMV
A- Retrieve resources using metadata and value matches	7	1	-	0.86	1.00	1.00	0.92
B- Specify Instances filtering by property value matches	4	1	-	0.75	1.00	1.00	0.86
C- Join of instances of different classes using metadata and value matches	15	15	6	0.00	0.60	0.60	0.60
D- Join a pair of instances of different classes using value matches	3	3	2	0.00	0.33	0.33	0.33
E- Join two same class resources to resources of another class	6	6	1	0.00	0.83	0.83	0.83
F- Join resources of various classes	5	5	1	0.00	0.80	0.80	0.80
OVERALL	40	31	10	0.27	0.76	0.76	0.72

BL – baseline tool KMV – KMV-synopses tool

AP- Average Precision

RR- Reciprocal Rank

Effectiveness. Table 4 summarizes the results for Mondial. Reading the table from left to right, the baseline tool failed in 10 queries (42% of the queries), while the KMV-synopses tool only failed in 1 of 24 queries (4% of the queries) w.r.t. the benchmark. Indeed, in Query 22 {*Atacama, Province, Argentina*}, the KMV-synopses tool failed because it chose the instance *Atacama* of class *Province* and tried to connect it with the instance *Argentina* of class *Country*, which resulted in an empty set of answers. For all query groups, the KMV-synopses tool achieved better average precision than the baseline tool. For instance, the baseline tool failed for query groups C and D, while the KMV-synopses tool processed all such queries correctly.

Table 5 summarizes the results for IMDB. Again, reading the table from left to right, the baseline tool failed in 31 queries (77.5% of the queries), while the KMV-synopses tool only failed in 10 of 40 queries (25% of the queries) w.r.t. the benchmark. The baseline tool failed for all IMDB benchmark queries in groups C to F since the use of the RDF schema for compiling the queries in these groups is not sufficient. Thus, the KMV-synopses tool reached a much higher mean average precision than the baseline tool. However, in Query 17 {*rick, blaine, movie*}, it failed since the keywords “*rick*”

and “*blaine*”, referring to instances of class *Person*, instead of instances of class
995 *Character*, joined with class *Movie*, generated a query that returned an empty set of
answers. Indeed, *Rick Blaine* did not work in movies, but rather in video movies.

Moreover, in Queries 28 {*tom, hanks, 2004*} and 29 {*audrey, hepburn, 1951*}, the
KMV-synopses tool matched the numbers “2004” and “1951” with values of properties
:death_date and :birth_date, respectively; the generated query then returned an empty set
1000 of answers, since these values do not simultaneously occur in instances that refer to
“*tom hanks*” and “*audrey hepburn*”.

Tables 4 and 5 also show the computed values of *Top-1* and *MRR* metrics for the
proposed algorithm. In both datasets, the obtained values are considerably higher. This
means that the proposed algorithm returned relevant top-1 answers for the non-failing
1005 queries.

Note that each line of these tables indicates the results and averages for a specific
query group; only the last line indicates the overall averages.

To summarize, the experiments show that the KMV-synopses tool outperforms the
baseline tool in all metrics.

1010 **Efficiency.** The *total elapsed time* depends on the *translation time*, that is, the time
to compile the keyword query into a SPARQL query, and the *execution time*, that is,
the time the RDF Search Engine takes to execute the SPARQL query. The total elapsed
time naturally depends on the RDF Search Engine chosen (Jena, in this case). Hence,
1015 we concentrate on the translation time.

In the KMV-synopses tool, the translation time can be broken in two components:
the *match time*, that is, the time it takes to match keywords with literals; and the
assembly time, that is, the time it takes to select the best matches and to discover how
to join the match results. The experiments indicated that, on average, the assembly time
1020 is 80% of the translation time.

The min time, max time, and average time, in seconds, for Mondial were 0.3s, 0.9s,
and 0.6s, respectively, and for IMDb were 1.2s, 59.3s, and 11.4s, respectively. The total
elapsed time was much higher for IMDb than for Mondial, since IMDb is 3 orders of
magnitude larger than Mondial, and since the keyword queries for IMDb were
1025 somewhat more complex.

The last columns of Table A.1 and Table A.2 (in Appendix), labeled with τ , show the total elapsed time the KMV-synopses tool took to execute each keyword-based query in the benchmark. Overall, the elapsed times of the KMV-synopses tool and the baseline tool were similar.

1030 From the experiments, for the KMV-synopses tool, we also observed that the translation time for Mondial was 45–52% of the total elapsed time, on average. For IMDb, it raised to 62–70% on average. This behavior can be explained because keyword matching is a costly process, which is heavily affected by the dataset size and the ambiguity of data, such as IMDb.

1035 Finally, we note that, for the KMV-synopses tool, when the keyword-based query had few matches, but the compiled SPARQL query had many joins, then the execution time represented most of the total elapsed time, such as for the Mondial benchmark keyword queries in Group *D*.

1040 **6. Comparison with Keyword search systems based on the “virtual documents” approach**

This section describes a second set of experiments that compares the schema-less approach proposed in this article with the state-of-the-art TSA+BM25 and TSA+VDP keyword search systems over RDF datasets based on the “virtual documents” approach [12], adopted as baselines. The experiments are based on the same datasets and queries adopted in [12].

6.1 Benchmark

This section summarizes the benchmark described in [12], for which we refer the reader. The benchmark and the code required to run the experiments are available at <https://bitbucket.org/account/user/keywordsearchrdfproject/projects/TSAC>.

1050 The original benchmark contains three real datasets – LinkedMDB, IMDb, and a subset of DBpedia – and two synthetic databases – the Lehigh University Benchmark

(LUBM) and the Berlin SPARQL Benchmark (BSBM). However, we decided not to use the LinkedMDB dataset since the number of triples is not significantly large. So, we selected the LUBM^{***}, BSBM^{†††}, IMDb^{‡‡‡}, and DBpedia^{§§§} datasets.

1055 LUBM is a database about universities, professors, and students developed by Lehigh University to facilitate the evaluation of Semantic Web Repositories. BSBM is a database built on an e-commerce use case, where different vendors with posted reviews offer a set of products. The LUBM benchmark has 14 SPARQL test queries^{****}, whereas the BSBM Explore use case^{††††} has 13 different SPARQL queries. For each
1060 synthetic dataset, a version of about 10M triples was used.

IMDb is a relational dataset that describes has movies, series, and artists are their relationships. We convert it into an RDF dataset with 256M triples. Finally, for DBpedia, we built an RDF graph composed by the triples in
1065 <<https://wiki.DBpedia.org/data-set-37>> corresponding to the DBpedia Ontology, the Ontology Infobox Types, the Titles subset, the Short Abstract subset, and the Raw Infobox Properties subset.

For IMDb, Dosso and Silvello designed 100 topics, where half were used for training and a half for testing. We used the 50 queries that they designed for testing. As for DBpedia, Dosso and Silvello considered 50 topics from QALD2_{te} and QALD2_{tr}, used
1070 by [3], manually mapped into CONSTRUCT SPARQL queries and the corresponding keyword queries. A topic is composed of three fields: the title, the dsc (description), and the SPARQL query. The SPARQL query contains the SPARQL query that returns the “correct” answer.

*** <http://swat.cse.lehigh.edu/projects/lubm/>

††† <http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/>

‡‡‡ <https://datasets.imdbws.com/>

§§§ <https://wiki.DBpedia.org/data-set-37>

**** <http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt>

†††† <http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/ExploreUseCase/index.html>

We computed the InfoRank values for resources, properties, and classes for all datasets and added them to the datasets. Table 6 summarizes the statistics about the used datasets. For comparison purposes, the benchmark is openly available at <https://figshare.com/s/d65d6a4ec70f169b4c50>.

Table 6 - Statistics – LUBM, BSBM, IMDb, and DBpedia datasets.

Dataset	Type	#Triples	#InfoRank Triples	#Queries
BSBM	synthetic	12M	1.6M	13
LUBM	synthetic	12M	1.7M	14
IMDb	real	256M	40.2M	50
DBpedia	real	72M	3.2M	50

6.2 Experimental Setup

KMV-synopses RDF keyword search tool. The KMV-synopses tool was already described in Section 5.2.

We computed the synopses for all datasets varying the parameter k . Table 7 shows the total space (in MB) required to store the KMV-synopses and the creation time in minutes. For the running experiment, we used $k=8,192$ for all datasets and tests.

Table 7 - KMV-synopses sizes (in MB) and creation time consumption (minutes)

Dataset	$k = 4,096$	$k = 8,192$	$k = 32,768$	Time
BSBM	2.83	4.66	13.3	~ 30
LUBM	1.26	2.47	9.12	~ 30
IMDb	0.89	1.96	7.21	~ 1000
DBpedia	109	143	232	~ 200

For DBpedia, we computed the KMV-synopses for 44,809 properties, considering their domains and ranges. Nevertheless, the sets of property domains and object property ranges are very skewed. For instance, we had:

- 607 domain synopses with $k = 8,192$
- 37 range synopses with $k = 8,192$
- 28,036 domain synopses with $k < 10$
- 8,138 range synopses with $k < 10$

For example, the object property `dbp:teamDirector` links a unique pair of resources: `dbr:Germany_national_handball_team` (in the domain) and `dbr:Tom_Schneider` (in the range).

1095 **Baseline.** As baseline for these experiments, we adopted the TSA+BM25 and the
TSA+VDP keyword search systems over RDF datasets based on the ‘‘virtual
documents’’ approach, described in [12]. These systems move most of the
computational complexity off-line and then exploit highly efficient text retrieval
1100 techniques and data structures to carry out the on-line phase. Dosso and Silvello showed
that these approaches are more efficient and effective, when compared with state-of-
the-art systems.

Metrics. The following definitions were introduced in [12] and are required to evaluate
the proposed approach.

1105 Recall that a *Cranfield framework* is a triple $C = (D, T, GT)$, where D is a dataset, T
is a set of topics, and GT defines the ground truths. In the context [12], D is an RDF
dataset, T is a set of keyword queries, and the *ground truth* for a topic $t_k \in T$, denoted
 G_{t_k} , is an RDF graph defined by the result of applying the SPARQL query
CONSTRUCT Q_{t_k} over D . The *relevant triples* for topic t_k are the triples that
1110 correspond to G_{t_k} ; all other triples are *not relevant* for topic t_k .

Let t_k be a topic, Q_k be the corresponding keyword query, and G_{t_k} be the ground truth
defined for t_k .

We assume that the keyword query system returns a *ranked answer list* $R_k = [ap_1,$
 $ap_2, \dots, ap_n]$, where each $ap_i = (G_i, sim_i)$ is an *answer pair*, where G_i is the *answer graph*
1115 at position i and sim_i is the *similarity* between G_i and G_{t_k} . As an abuse of notation, we
write $G_i \in R_k$, when there is an answer pair of the form (G_i, sim_i) in R_k .

The *Signal-to-Noise Ratio* (SNR) of G_i is defined as

$$SNR(G_i) = \frac{|(G_i \cap G_{t_k}) - S|}{|G_i|} \quad (30)$$

where S is the union of all relevant triples for topic t_k that are also in the answer graph
at position j , for all $j \in [1, i]$.

1120 Note that SNR represents a kind of precision score, where the numerator is equal to the number of relevant triples in G_i found for the first, that is, excluding the triples in answer graphs that preceded G_i in the ranking R_k , and the denominator is the total number of triples in G_i . Example 2, taken from [12], illustrates the definition of SNR.

Let λ be a *relevance parameter* such that an answer graph $G_i \in R_k$ is considered
1125 *relevant* iff $SNR(G_i) > \lambda$.

The *recall* of a ranked answer list R_k for topic t_k ground truth G_{t_k} is defined as

$$recall(R_k) = \frac{|\bigcup_{G_i \in R_k | SNR(G_i) \geq \lambda} (G_i \cap G_{t_k})|}{|G_{t_k}|} \quad (31)$$

Intuitively, the recall of R_k is the ratio between the set of relevant triples that appear in some relevant answer graph in R_k and the cardinality of the ground truth.

The *precision* of a ranked answer list R_k for topic t_k ground truth G_{t_k} is defined as

$$precision(R_k) = \frac{|\bigcup_{G_i \in R_k | SNR(G_i) \geq \lambda} (G_i \cap G_{t_k})|}{|\bigcup_{G_i \in R_k} G_i|} \quad (32)$$

1130 Intuitively, the precision of R_k is the ratio between the set of relevant triples that appear in some relevant answer graph in R_k and the cardinality of the set of triples that appear in some answer graph in R_k .

The *precision at c*, denoted $prec@c$, is the precision computed considering the first c elements of the ranking, and is defined as

$$prec@c = \frac{|\bigcup_{G_i \in R_k | SNR(G_i) \geq \lambda \wedge i \in [1,c]} (G_i \cap G_{t_k})|}{|\bigcup_{G_i \in R_k | i \in [1,c]} G_i|} \quad (33)$$

1135 The *Graph Relevance Weight* (GRW) measures the relevance of the answer graph at i of ranking R_k :

$$GRW(G_i) = \frac{|(G_i \cap G_{t_k}) - S|}{|G_{t_k}|} \quad (34)$$

where S again is the union of all relevant triples for topic t_k that are also in the answer graph at position j , for all $j \in [1, i]$.

1140 The *Relevance Gain* (RG) computes the relevance gain of an answer G_i in ranking R_k of size n ($n > 0$), for a given topic t_k , considering $\lambda \in [0, 0.1, 0.2, \dots, 1]$ and a position b ($b > 0$).

$$RG_b(G_i) = \begin{cases} GRW(G_i) & \text{if } i \leq b \wedge SNR(G_i) > \lambda \\ \frac{GRW(G_i)}{\log_b i} & \text{if } i > b \wedge SNR(G_i) > \lambda \\ 0 & \text{if } SNR(G_i) \leq \lambda \end{cases} \quad (35)$$

Finally, the *triple-based Discounted Cumulative Gain (tb-DCG)* of a ranking R_k is defined as:

$$tb-DCG_b(R_k) = \sum_{i=1}^n RG_b(G_i) \quad (36)$$

This metric measures the overall utility of a subgraph ranking for the end-users. It weighs the top-heaviness (best answers are ranked first) and essentialness (absence of redundancy) in the ranking.

Hardware and software setup. All tests were performed under the same conditions described in Section 5.2.

1150 6.3 Experimental Evaluation

This section describes the results obtained for the KMV-synopses tool and compares them with the baseline results from [12], using the metrics summarized in Section 6.2, also from [12].

1155 **Effectiveness.** Table 8 shows the values of the metrics computed for the KMV-synopses tool, along with the values for the baselines from [12] over the four datasets. To compare with the baselines, we used $\lambda=0$ to estimate the relevance of the answer graphs in the rankings returned by the KMV-synopses tool, as in [12]. Recall that, by choosing $\lambda=0$, every answer containing at least one relevant triple is considered
1160 relevant.

Table 8 - Results obtained with the experiments using *SRR* and $\lambda = 0$.

Dataset	System	Prec@1	Prec@5	Recall	tb-DCG
BSBM	TSA+BM25	0.039 ± 0.01	0.010 ± 0.00	0.227 ± 0.07	0.139 ± 0.05
	TSA+VDP	0.071 ± 0.03	0.071 ± 0.03	0.047 ± 0.03	0.074 ± 0.03
	KMV-Synopses	0.815	0.823	0.852	0.720
LUBM	TSA+BM25	0.082 ± 0.04	0.111 ± 0.05	0.505 ± 0.07	0.281 ± 0.07
	TSA+VDP	0.145 ± 0.03	0.226 ± 0.05	0.384 ± 0.03	0.234 ± 0.06
	KMV-Synopses	0.905	0.885	0.684	0.539
IMDb	TSA+BM25	0.011 ± 0.00	0.009 ± 0.00	0.273 ± 0.36	0.067 ± 0.01
	TSA+VDP	0.006 ± 0.00	0.006 ± 0.00	0.363 ± 0.04	0.308 ± 0.04
	KMV-Synopses	0.810	0.761	0.648	0.681
DBpedia	TSA+BM25	0.000 ± 0.00	0.000 ± 0.00	0.851 ± 0.03	0.135 ± 0.01
	TSA+VDP	0.002 ± 0.00	0.002 ± 0.00	0.129 ± 0.03	0.118 ± 0.03
	KMV-Synopses	0.233	0.217	0.191	0.363

For the BSBM, LUBM, and IMDb datasets, the KMV-synopses tool obtained higher metrics values than the TSA systems based on the “virtual document” approaches. As for precision, this means that the KMV-synopses tool finds a larger number of relevant triples w.r.t the ground truth and ranks them adequately, contrasting with the TSA systems, that return a high number of noisy triples in the answers.

Focusing on DBpedia, the “virtual document” approach based on the BM25 function had a higher recall value than the other systems but obtains 0 for precision values. This means that the system returns many relevant triples but cannot rank them effectively. Moving to precision and tb-DCG, the KMV-synopses tool obtained better values, even though the tool failed in 19 of the 50 benchmark queries (the synthesized SPARQL queries returned empty answers). This fact also influenced the low recall value. Another factor that affects the effectiveness of the KMV-synopses tool is the high degree of ambiguity of DBpedia. For example, the keyword “governor” exactly matches the rdfs:label properties of class dbo:Governor and the object property dbo:governor. Thus, deciding which element should be used to synthesize the SPARQL query is critical for the KMV-synopses tool. The current implementation prioritizes the class match found, as explained in Section 4.3. In the future, it is advisable to improve this heuristic by

analyzing the keyword-based query context (for example, the sequence of keywords)
1180 to enhance precision and to return relevant answer graphs.

Concerning IMDb, the dataset also has a high degree of ambiguity but, in this case,
the ambiguity has to do with the resource property values. For instance, the keyword
“*Will Smith*” matches more than a hundred property values. However, using the ranking
1185 heuristic described in Section 4.3, the KMV-synopses tool selected and included in the
synthesized SPARQL query the resource with the highest *InfoRank* value, in this case,
the resource expected in the ground truth. This fact again raises the discussion if a
manually defined ground truth covers all possible answers for a keyword-based query.

As for LUBM, the KMV-synopses tool synthesized SPARQL queries with non-
empty answers for all topics in the benchmark. However, there are non-relevant graphs
1190 in the answers, since the precision value is not 1. For example, the system does not
achieve perfect precision for the benchmark query $Q2 = \{GraduateStudent, University,$
 $Department, memberOf, subOrganizationOf, undergraduateDegreeFrom\}$, where the
ground truth is the graph resulting from the CONSTRUCT SPARQL query (query
details were omitted by brevity):

```
1  CONSTRUCT WHERE{  
2   ?X rdf:type swat:GraduateStudent .  
3   ?Y rdf:type swat:University .  
4   ?Z rdf:type swat:Department .  
5   ?X swat:memberOf ?Z .  
6   ?Z swat:subOrganizationOf ?Y .  
7   ?X swat:undergraduateDegreeFrom ?Y }
```

1195 Note that the variable *?Y* binds resources of class *swat:University* (line 3) and then *?Y*
simultaneously appears in the object of the triple patterns of the object properties
swat:subOrganizationOf and *swat:undergraduateDegreeFrom* (lines 6 and 7). The WHERE
clause indicates a triangular pattern of relationships between the objects involved,
which is hard to indicate through keywords. Thus, the KMV-synopses tool compiles a
1200 SPARQL query similar to above, replacing the variable *?Y* by *?V* in line 7.

A similar situation was observed for some queries in the IMDb and DBpedia
benchmarks. For example, the IMDb benchmark queries Q31 to Q40 and the DBpedia

benchmark queries Q1 and Q21 find films where a person (identified by her/his name) is an actress/actor and, at the same time, the film is directed/written/produced by herself/himself, such as the benchmark query $Q1=\{Clint\ Eastwood, starring, director\}$ in DBpedia.

To summarize, the experiments showed that the KMV-synopses tool obtained good precision and recall values, and also produced reasonable rankings. Indeed, the KMV-synopses tool compared favorably with the baseline, state-of-art systems in terms of effectiveness.

Finally, we observe that, unlike [12], the synthetic databases posed no problems for the KMV-synopses tool, whereas IMDb and DBpedia were harder to handle, due to their ambiguity. Furthermore, recall from Section 6.2 that [12] defined the ground truth for each keyword query as a single RDF graph that is the result of a manually specified CONSTRUCT SPARQL query. This decision raises at least two questions when assessing the effectiveness of an RDF keyword-based query system. First, the SPARQL query does not necessarily cover all possible answers for the keyword query. Second, the answers are not individualized, so computing effectiveness required redefining the notions of precision and recall, as summarized in Section 6.2.

Efficiency. Recall the translation time corresponds to the time to compile the keyword query into a SPARQL query and the total elapsed time is the time consumed by the system since it receives the keyword query until it returns the corresponding answer. We consider that comparing the times of the KMV-synopses system against the on-line times reported for the baselines in [12] is not reasonable since the experimental environments and RDF engines used in both experiments were different. However, we included, in Table 9, the translation and the total elapsed times of our proposed system for all four datasets, which are comparable to those reported in in [12] for the baselines.

Table 9 - Min, Max, and Average for Translation and Total Elapsed Times (in sec).

Dataset	Translation Time			Total Elapsed Time		
	Min	Max	Ave	Min	Max	Ave
BSBM	2.468	9.768	6.195	4.529	11.280	7.322
LUBM	4.213	8.776	6.205	5.188	9.120	7.609
IMDb	1.542	102.792	60.609	2.896	1200	254.524
DBpedia	0.896	560.541	95.227	1.654	1200	441.875

Concerning the execution times, as expected, the times consumed to run the queries
 1230 on top of the synthetic datasets – the BSBM and LUBM datasets – were considerably
 faster than the times in the real datasets – the IMDb and DBpedia datasets.

We focus here on the *translation time* of the KMV-synopses tool. We observed that,
 on average, the translation times for the benchmark queries of the BSBM and LUBM
 datasets were very similar, which can be explained by the low degree of ambiguity,
 1235 which in turn implies a few matches for a keyword in the query.

Moving to IMDb, the translation time for the benchmark queries was also reasonably
 low. The dataset structure explains this behavior since the IMDb dataset essentially
 consists of instances of two different classes (Person and Film) with their datatype
 properties and few object properties linking them. So, the assembly process does not
 1240 consume much time connecting the resources resulting from the matching process. On
 average, the execution time of a query represented 80% of the total elapsed time. In
 some cases, this proportion even exceeded 98%.

Regarding DBpedia, the performance of the KMV-Synopses tool was much lower
 in terms of translation times. This fact was expected because, as mentioned, the degree
 1245 of ambiguity and the graph structure, regarding the number of object properties
 available to connect the resources in the graph, influences the matching and assembly
 processes, respectively. On average, the translation time consumed 76% of the total
 time. Concerning the ratio between the matching and the assembly processes times, we
 observed that, on average, the assembly process consumed more time, except for the
 1250 typical cases described at the end of Section 4.4, since the assembly process directly
 compiles the described triple pattern.

6.4 Effectiveness using an Alternative Measure for Graph Relevance

As mentioned in Section 6.2, Dosso and Silvello [12] defined *SNR* to determine when an answer graph is relevant. However, this measure strongly considers the relevant
1255 triples observed at the top-ranked graphs. By contrast, the *KMV-synopses* tool returns answer graphs with relevant triples in any ranking position. Thus, this section proposes a different measure to establish when an RDF graph is relevant. It is based on the number of relevant and non-relevant triples in the RDF graph, but it punishes the presence of non-relevant triples, and does not memorize the relevant triples in previous
1260 rank positions.

Given a topic $t_k \in T$, a ranking R_k , and the ground truth graph G_{t_k} , we define the *Graph Relevance Ratio (GRR)* of $G_i \in R_k$ to establish when a graph is relevant. Note that this measure is asymmetric, as we are only interested in comparing G_i against G_{t_k} . Hence, we use a variant of the Tversky index [39].

$$S(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha |X - Y| + \beta |Y - X|} \quad (37)$$

1265 From Eq. 37, if we consider S as *GRR*, $X = G_i$, $Y = G_{t_k}$, $\alpha = 1$, and $\beta = 0$, then the *GRR*(G_i, G_{t_k}), or simply *GRR*(G_i), is defined as

$$GRR(G_i) = \frac{|G_i \cap G_{t_k}|}{|G_i \cap G_{t_k}| + |G_i - G_{t_k}|} \quad (38)$$

Intuitively, by taking $\alpha = 1$, and $\beta = 0$, we consider the triples in the answer graph G_i which are not in the ground truth G_{t_k} , and ignore the triples which are in the ground truth G_{t_k} , but not in the answer graph G_i . Note that Eq. 38 is then equivalent to set
1270 containment, since $|G_i \cap G_{t_k}| + |G_i - G_{t_k}| = |G_i|$. Indeed, we can redefine *GRR* as

$$GRR(G_i) = \frac{|G_i \cap G_{t_k}|}{|G_i|} \quad (39)$$

Also, note that the *RR* of an answer graph G_i , as *SNR*, rewards precise and essential graphs as it decreases whenever the graph contains non-relevant triples.

Now, inspired by [12], we redefine *precision*(R_k) and *precision@c*(R_k), using *GRR*, as

$$precision(R_k) = \frac{|\{G_i \in R_k | GRR(G_i) \geq \lambda\}|}{|R_k|} \quad (40)$$

$$prec@c(R_k) = \frac{|\{G_i \in R_k | GRR(G_i) \geq \lambda \wedge i \in [1, c]\}|}{c} \quad (41)$$

1275 By requiring that $GRR(G_i) \geq \lambda$, we discard those answer graphs G_i that have fewer relevant triples in the ground truth, as compared with the total number of triples in G_i .

Here, *precision* is the ratio between the total number of relevant graphs and the total number of graphs in the ranking.

1280 However, we decided not to name *recall* the metric equivalent to that proposed in [12], since the ground truth for a topic t_k , G_{t_k} , is a compact graph that does not individualize the answers. Therefore, we redefine the recall metric of Section 6.2 under the name *Relevant Triples Ratio* of R_k ($RTR(R_k)$, for short) as

$$RTR(R_k) = \frac{|\cup_{G_i \in R_k | RRR(G_i) \geq \lambda} (G_i \cap G_{t_k})|}{|G_{t_k}|} \quad (42)$$

1285 We then computed the *precision*, *prec@1*, *prec@5*, and *RTR* values for the results of the experiments with the four datasets, using all values of λ in the set $\{0.0, 0.1, 0.2, \dots, 1.0\}$, and Equations 40, 41, and 42. Figure 7a shows the values for BSBM, Figure 7b for LUBM, Figure 7c for IMDb, and Figure 7d for DBpedia.

1290 Note that, as expected, when the value of λ increases, the metrics values decrease. It means that the more restrictive it is to consider that an answer graph is relevant, fewer of the answers found are regarded as “correct” w.r.t G_{t_k} . As mentioned, the KMV-synopses tool returns non-empty answers for all benchmark queries in the LUBM dataset, and for relevance parameter $\lambda \leq 0.5$, the precision value is perfect. It means that, for each topic, all returned graphs are relevant. We observed that the *RTR* values are quite stable for all values of λ . It means that the answer graphs considered as non-

relevant contain few relevant triples. We note that, for $\lambda = 0.5$ or larger, the metrics values decreased. For the BSBM dataset, they begin to decrease at $\lambda = 0.8$.

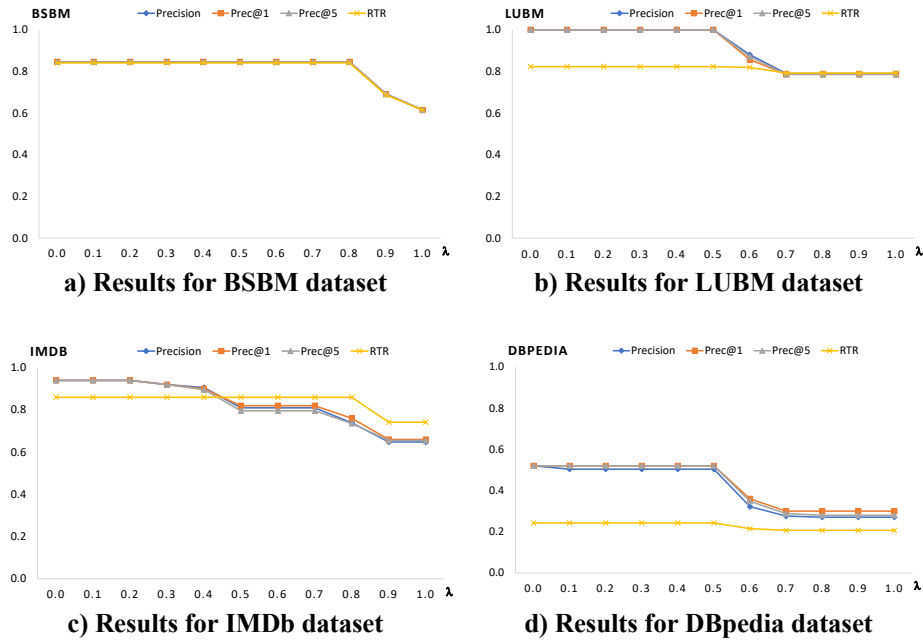


Figure 7. Metrics values computed for the four datasets.

Therefore, Table 10 shows only the metrics values obtained with the relevance parameter $\lambda = 0.8$ for all four datasets.

Table 10. Metrics values computed using the proposed *SRR* and $\lambda = 0.8$

Dataset	Precision	Prec@1	Prec@5	RTR
BSBM	0.846	0.846	0.846	0.841
LUBM	0.790	0.786	0.786	0.792
IMDb	0.739	0.760	0.736	0.860
DBpedia	0.271	0.300	0.280	0.207

Again, computing measures as DCG or NDCG is not viable since, as already mentioned, the ground truth is not a ranked list of individualized answers but a single RDF graph.

7. Conclusions and Future Work

This article addressed the problem of implementing keyword search for RDF datasets that do not necessarily feature an RDF schema. It introduced a novel algorithm to automatically translate a user-specified keyword-based query into a SPARQL query that returns answers with respect to the keywords. The algorithm synthesizes the SPARQL query by exploring the Jaccard and set containment similarity measures between the property domains and ranges and class instance sets, observed in the RDF dataset. The algorithm estimates these similarity measures using KMV-synopses, which can be pre-computed in a single pass over the RDF dataset.

The article then describes two sets of experiments with an implementation of the proposed algorithm, which we called the KMV-synopses RDF keyword search tool, or simply the KMV-synopses tool.

The first set of experiments compared the KMV-synopses tool with a baseline schema-based tool [14] over a benchmark. We were interested in testing if schema information could be replaced by KMV-synopses, without impacting performance. The experiments showed that the KMV-synopses tool outperformed the baseline tool in all metrics adopted, which shows that the lack of schema information can indeed be replaced by pre-computed, concise KMV-synopses for the property domains and ranges and class instance sets. Also, the average elapsed times of the baseline tool and the KMV-synopses tool were similar, which indicates that estimating set similarity based on KMV-synopses does not introduce significant overhead, even for large RDF datasets such as IMDb, if the KMV-synopses are pre-computed.

The second set of experiments indicated that the KMV-synopses tool performed better than the state-of-the-art TSA+BM25 and TSA+VDP keyword search systems over RDF datasets based on the “virtual documents” approach, using the metrics and the benchmarks proposed originally to assess these systems.

Finally, we proposed the *Graph Relevance Ratio (GRR)* to establish when an answer graph is relevant w.r.t. a ground truth graph. It is based on the number of relevant and non-relevant triples in the RDF graph, but it punishes the presence of non-relevant triples, and does not memorize the relevant triples in previous rank positions.

We are currently working on an index for the KMV-synopses to optimize the KMV-synopses tool. We are also investigating methods to maintain KMV-synopses incrementally [24].

Finally, the strategy described in this article can be combined with schema information to drive the query compilation process. The KMV-synopses will be used to help the query compilation process in much the same way as the usual database statistics help the query optimization process. We also plan to adapt our earlier keyword search tool [23], which is schema-based and operates over relational and RDF datasets, along these lines.

Acknowledgements

This work was partly funded by grants CAPES/88881.134081/2016-01, CNPq/302303/2017-0, and FAPERJ/E-26-202.818/2017 and E-26/200.770/2019. Carlos Oliveira was partially supported by the Project CEMAPRE/REM - UIDB/05069/2020 - financed by FCT/MCTES through national funds. The authors also wish to thank Dennis Dosso and Gianmaria Silvello for their invaluable help with the benchmarks used in Section 6.

References

- [1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, S. Sudarshan. BANKS: Browsing and keyword searching in relational databases. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB '02)*, Hong Kong SAR, China, 20–23 August 2002, pp. 1083-1086. DOI: 10.1016/B978-155860869-6/50114-1
- [2] S. Agrawal, S. Chaudhuri, G. Das. DBXplorer: A system for keyword-based search over relational databases. In *Proceedings of the 18th International*

- Conference on Data Engineering (ICDE '02)*, San Jose, CA, USA, 26 Feb.-1 March 2002. pp. 5-16. DOI: 10.1109/ICDE.2002.994693
- 1360 [3] K. Balog and R. Neumayer. A test collection for entity search in DBpedia. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland (July, 2013), pp. 737–740. DOI: <https://doi.org/10.1145/2484028.2484165>
- 1365 [4] H. Bast, B. Buchhold, E. Haussmann. Semantic search on text and knowledge bases. *Found. and Trends® in Info. Retr.*, 10(1), (2016), 119-271. DOI: 10.1561/1500000032
- 1370 [5] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo-Lado, Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data - SIGMOD '11*, June 2011, Pages 565–576. DOI: <https://doi.org/10.1145/1989323.1989383>
- 1375 [6] S. Bergamaschi, F. Guerra, M. Interlandi, R. Trillo-Lado, Y. Velegrakis. Combining user and database perspective for solving keyword queries over relational databases. *Inf. Syst.* 55, C (Jan. 2016), 1-19. DOI: 10.1016/j.is.2015.07.005
- 1380 [7] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, R. Gemulla. On synopses for distinct-value estimation under multiset operations. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD '07)*. ACM, New York, NY, USA, pp. 199-210. DOI: 10.1145/1247480.1247504
- [8] C. Bizer and A. Schultz, The Berlin SPARQL benchmark, *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 2 (2009), 1–24.
- [9] D. Brickley, R. V. Guha (eds). 2014. RDF Schema 1.1. W3C Recommendation (25 February 2014).
- 1385 [10] J. Coffman, A. C. Weaver. A framework for evaluating database keyword search strategies. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*. ACM, New York, NY, USA, pp. 729-738. DOI: 10.1145/1871437.1871531

- 1390 [11] R. Cyganiak, D. Wood, M. Lanthaler (eds.). *RDF 1.1 Concepts and Abstract Syntax*. W3C Recommendation (25 Feb. 2014).
- [12] D. Dosso and G. Silvello. Search Text to Retrieve Graphs: A Scalable RDF Keyword-Based Search System, *IEEE Access*, vol. 8, pp. 14089-14111 (2020). DOI: 10.1109/ACCESS.2020.2966823.
- 1395 [13] S. Elbassuoni, R. Blanco. Keyword search over RDF graphs. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM '11)*, ACM, New York, NY, USA, pp. 237-242. DOI: 10.1145/2063576.2063615
- 1400 [14] G. M. García, Y. T. Izquierdo, E. Menendez, F. Dartayre, M. A. Casanova. RDF Keyword-based Query Technology Meets a Real-World Dataset. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT '17)*, March 21-24, 2017 - Venice, Italy: ISBN 978-3-89318-073-8, on OpenProceedings.org, pp. 656-667. DOI: 10.5441/002/edbt.2017.86
- 1405 [15] A. Ghanbarpour, H. Naderi. A Model-based Keyword Search Approach for Detecting Top-k Effective Answers. *The Computer Journal*, 62(3), March 2019, 377–393. DOI: 10.1093/comjnl/bxy056
- 1410 [16] K. Gkirtzou, K. Karozos, V. Vassalos, T. Dalamagas, Keywords-to-SPARQL translation for RDF data search and exploration. In *Proceedings of the 19th International Conference on Theory and Practice of Digital Libraries – TPDL 2015*, Poznan, Poland (Sept. 14-18, 2015), pp. 111-123. DOI: 10.1007/978-3-319-24592-8_9
- [17] Y. Guo, Z. Pan, and J. Heflin, LUBM: A benchmark for OWL knowledge base systems, *J. Web Semantics* 3(2–3) (Oct. 2005), 158–182. DOI: 10.1016/j.websem.2005.06.005
- 1415 [18] M. Hadjieleftheriou, X. Yu, N. Koudas, D. Srivastava. Hashed Samples: Selectivity Estimators for Set Similarity Selection Queries. In *Proceedings of the VLDB Endow.* 1, 1 (August 2008), 201-212. DOI: 10.14778/1453856.1453883

- 1420 [19] S. Han, L. Zou, X. Yu, D. Zhao. Keyword Search on RDF Graphs - A Query Graph Assembly Approach. In *Proceedings of the 2017 ACM Conference on Information and Knowledge Management (CIKM '17)*. ACM, New York, NY, USA, 227-236. DOI: 10.1145/3132847.3132957
- [20] S. Harris, A. Seaborne. *SPARQL 1.1 Query Language*. W3C Recommendation (21 Mar. 2013).
- 1425 [21] H. He, H. Wang, J. Yang, P. S. Yu. Blinks: Ranked keyword searches on graphs. In *Proceedings 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD '07)*. ACM, New York, NY, USA, pp. 305-316. DOI: 10.1145/1247480.1247516
- 1430 [22] V. Hristidis, Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB '02)*, Hong Kong SAR, China, 20–23 August 2002, pp. 670-681. DOI: 10.1016/B978-155860869-6/50065-2
- 1435 [23] Y. T. Izquierdo, G. M. García, E. S. Menendez, M. A. Casanova, F. Dartayre, C. Levy. QUIOW: A Keyword-Based Query Processing Tool for RDF Datasets and Relational Databases. In: *Database and Expert Systems Applications (DEXA 2018)*. Lecture Notes in Computer Science, vol. 11030. Springer, Cham (2018), pp. 259-269. DOI: 10.1007/978-3-319-98812-2_22
- 1440 [24] Y. T. Izquierdo, Keyword Search Algorithm over Large RDF Datasets. In: Guizzardi G., Gailly F., Suzana Pitangueira Maciel R. (eds) *Advances in Conceptual Modeling. ER 2019*. Lecture Notes in Computer Science, vol 11787. Springer, Cham. DOI: 10.1007/978-3-030-34146-6_21
- 1445 [25] W. Le, F. Li, A. Kementsietsidis, S. Duan. Scalable Keyword Search on Large RDF Data. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 26, Issue 11 (Nov. 2014), pp. 2774 - 2788. DOI: 10.1109/TKDE.2014.2302294
- [26] W. Le, S. Duan, A. Kementsietsidis, F. Li, M. Wang. 2011. Rewriting queries on SPARQL views. In *Proceedings of the 20th international conference on World wide web (WWW '11)*. ACM, New York, NY, USA, 655-664. DOI=<http://dx.doi.org/10.1145/1963405.196349>

- 1450 [27] X-Q. Lin, Z-M. Ma, L. Yan. RDF Keyword Search Using a Type-based
Summary. *Journal of Information Science and Engineering* 34, pp. 489-504
(2018) DOI: 10.6688/JISE.2018.34.2.11
- [28] Z. Ma, X. Lin, L. Yan, and Z. Zhao. RDF Keyword Search by Query
Computation. *Journal of Database Management* 29(4) (Sept. 2018), 1-27.
1455 DOI:10.4018/JDM.2018100101
- [29] E. S. Menendez, M. A. Casanova, L. A. P. Paes Leme, M.
Boughanem. Novel Node Importance Measures to Improve Keyword Search
over RDF Graphs. In *Proceedings of the 30th International Conference on
Database and Expert Systems Applications (DEXA 2019)*, Linz, Austria,
1460 August 26–29, 2019, Proceedings, Part II. Lecture Notes in Computer
Science, vol 11707, Springer Nature, pp. 143–158.
DOI: https://doi.org/10.1007/978-3-030-27618-8_11
- [30] E. S. Menendez. *Novel Node Importance Measures to Improve Keyword
Search over RDF Graphs*. Doctoral Thesis. Department of Informatics,
1465 PUC-Rio (February 2019). DOI: 10.17771/PUCRio.acad.37741
- [31] A. B. Neves Jr., L. A. P. Paes Leme, RDF Datasets to Test Keyword Search
Algorithms. DOI: <https://doi.org/10.6084/m9.figshare.11347676.v1>
- [32] A. B. Neves Jr., L. A. P. Paes Leme, Y. T. Izquierdo, G. M. García, M. A.
Casanova, Computing Benchmarks for RDF Keyword Search (submitted
1470 for publication).
- [33] B. P. Nunes, J. Herrera, D. Taibi, G. R. Lopes, M. A. Casanova, S. Dietze.
SCS Connector - Quantifying and Visualising Semantic Paths Between
Entity Pairs. In *Proceedings of the Satellite Events of the 11th European
Semantic Web Conference (ESWC'14)*. Springer, Heraklion, Greece, 461–
1475 466. DOI: 10.1007/978-3-319-11955-7_67
- [34] A. C. Oliveira Filho. Benchmark para Métodos de Consultas por Palavras-
Chave a Bancos de Dados Relacionais. Dissertação apresentada ao Programa
de Pós-Graduação do Instituto de Informática da Universidade Federal de
Goiás, 2018.

- 1480 [35] P. Oliveira, A. Silva, E. Moura. Ranking Candidate Networks of relations to
improve keyword search over relational databases. Proc. IEEE 31st
International Conference on Data Engineering (ICDE '15), Seoul, South
Korea, 2015 pp. 399-410. DOI: 10.1109/ICDE.2015.7113301 ICDE 2015,
399-410
- 1485 [36] M. S. Ramada, J. C. da Silva, P. S. Leitão-Júnior. From keywords to
relational database content: A semantic mapping method, *Inf. Syst.* 88 (2020)
101460. DOI: <https://doi.org/10.1016/j.is.2019.101460>
- [37] M. Rihany, Z. Kedad, S. Lopes. Keyword Search Over RDF Graphs Using
WordNet. In *Proceedings of the 1st Int'l. Conf. on Big Data and Cyber-*
1490 *Security Intelligence*. Hadath, Lebanon, Dec. 13-15, 2018.
- [38] T. Tran, H. Wang, S. Rudolph, P. Cimiano. Top-k exploration of query
candidates for efficient keyword search on graph-shaped (rdf) data. In
Proceedings 2009 IEEE International Conference on Data Engineering
(ICDE '09). IEEE Computer Society, Washington, DC, USA, pp. 405-416.
1495 DOI: 10.1109/ICDE.2009.119
- [39] A. Tversky. Features of similarity. *Psychological Review* 84(4) (1977), 327–
352. DOI: <https://doi.org/10.1037/0033-295X.84.4.327>
- [40] P. Venetis, Y. Sismanis, B. Reinwald. CRSI: A Compact Randomized
Similarity Index for Set-Valued Features. In *Proceedings of the 15th*
1500 *International Conference on Extending Database Technology (EDBT '12)*.
ACM, New York, NY, USA, PP. 384-395. DOI: 10.1145/2247596.2247642
- [41] 34 Vinay M.S. and J. R. Haritsa. Operator implementation of Result Set
Dependent KWS scoring functions, *Inf. Syst.* 89 (2020) 101465. DOI:
<https://doi.org/10.1016/j.is.2019.101465>
- 1505 [42] H. Wang, K. Zhang, Q. Liu, T. Tran, Y. Yu. Q2Semantic: A Lightweight
Keyword Interface to Semantic Search. In *The Semantic Web: Research and*
Applications. ESWC 2008. Lecture Notes in Computer Science, vol 5021.
Springer, Berlin, Heidelberg, pp 584-598. DOI: [https://doi.org/10.1007/978-](https://doi.org/10.1007/978-3-540-68234-9_43)
3-540-68234-9_43

- 1510 [43] Y. Wen, Y. Jin, X. Yuan. KAT: Keywords-to-SPARQL Translation Over
RDF Graphs. In *Database Systems for Advanced Applications (DASFAA
2018)*. Lecture Notes in Computer Science, vol 10827. Springer, Cham, pp
802-810. DOI: 10.1007/978-3-319-91452-7_51
- [44] Y. Yang, Y. Zhang, W. Zhang, Z. Huang. GB-KMV: An Augmented KMV
1515 Sketch for Approximate Containment Similarity Search.
arXiv:1809.00458v1
- [45] H. Yoghoudjian, S. Elbassuoni, M. Jaber, and H. Arnaout. Top-k Keyword
Search over Wikipedia-based RDF Knowledge Graphs. In *Proceedings of
the 9th International Joint Conference on Knowledge Discovery, Knowledge
1520 Engineering and Knowledge Management - Volume 1: KDIR*, pp 17-26.
DOI: 10.5220/0006411400170026
- [46] G. Zenz, X. Zhou, E. Minack, W. Siberski, W. Nejdl. From keywords to
semantic queries - incremental query construction on the semantic web. *Web
Semantics: Science, Services and Agents on the World Wide Web*, 7(3),
1525 (Sept. 2009), pp. 166-176. DOI: 10.1016/j.websem.2009.07.005
- [47] W. Zheng, L. Zou, W. Peng, X. Yan, S. Song, D. Zhao. Semantic SPARQL
similarity search over RDF knowledge graphs. In *Proceedings of the VLDB
Endow.* 9, 11 (July 2016), 840-851. DOI: 10.14778/2983200.2983201
- [48] Q. Zhou, C. Wang, M. Xiong, H. Wang, Y. Yu. SPARK: Adapting keyword
1530 query to semantic search. In *Proceedings of the 6th International Semantic
Web Conference and 2nd Asian Semantic Web Conference
(ISWC'07/ASWC'07)*. Springer-Verlag, Berlin, Heidelberg, 694-707. DOI:
10.1007/978-3-540-76298-0_5

Appendix

1535

Table A.1.

Query workload for Mondial and the Total Elapsed Time for each query.

Groups	Keyword Queries	τ
A) Retrieve instances of classes (using metadata and value matches)	1. niger country	0.6
	2. atacama desert	0.5
	3. mongolia parliamentary	0.5
	4. everest elevation	0.4
B) Retrieve joined instances of different classes (using metadata and value matches)	5. spain galician	0.5
	6. poland language	0.3
	7. haiti religion	0.4
	8. brazil brasilia	0.4
C) Retrieve joined instances of the same class (using value matches)	9. mongolia china	0.7
	10. lebanon syria	0.7
	11. mali france	0.8
	12. brazil portugal	0.5
D) Retrieve two instances from same class joined by instances of another class	13. poland cape verde organization	0.8
	14. iceland mali organization	0.7
	15. mauritius india organization	0.9
	16. vanuatu afghanistan organization	0.6
E) Retrieve two instances of different classes joined by elements of another class through intermediary nodes	17. hutu country africa	0.8
	18. country asia uzbek	0.8
	19. country america catholic	0.8
	20. country european jewish	0.9
F) Retrieve instances from various classes joined between them	21. paranaiba province brazil	0.6
	22. atacama province argentina	0.8
	23. everest province china	0.7
	24. rhein germany province	0.7

Table A.2.

Query workload for IMDb and the Total Elapsed Time for each query.

Groups	Keyword Queries	τ
A) Retrieve instances of classes (using metadata and value matches)	1. denzel washington person	3.5
	2. johnny depp actor	5.5
	3. forrest gump work	2.6
	4. star wars movie	13.7
	5. angelina jolie gender	5.2
	6. the sound of music length	5.2
	7. lord of the rings novel	7.5
B) Retrieve instances filtering by property value matches	8. will smith male	59.4
	9. tom hanks "9 july 1956"	33.4
	10. gone with the wind "august 1991"	20.7

	11. casablanca “they had a date with fate in casablanca”	46.4
C) Retrieve joined instances of different classes (using metadata and value matches)	12. johnny depp work	5.9
	13. morgan freeman work	4.2
	14. atticus finch movie	1.7
	15. indiana jones movie	8.5
	16. james bond movie	18.6
	17. rick blaine movie	1.8
	18. will kane movie	2.1
	19. dr. hannibal lecter movie	1.2
	20. norman bates movie	1.2
	21. darth vader movie	4.4
	22. the wicked witch of the west movie	2.1
	23. nurse ratched movie	1.5
	24. jacques clouseau actor	1.3
	25. jack ryan actor	2.6
	26. terminator actor	9.3
D) Retrieve two linked instances of different classes (using value matches)	27. clint eastwood frank harrigan	10.2
	28. tom hanks 2004	5.7
	29. audrey hepburn 1951	32.5
E) Retrieve two instances of the same class joined by elements of another class	30. julia roberts richard gere work	12.6
	31. harrison ford george lucas work	30.2
	32. sean connery ian fleming work	20.7
	33. keanu reeves lana wachowski work	6.3
	34. dean jones herbie	15.6
	35. Indiana Jones and the last crusade raiders of the lost ark person	5.4
F) Retrieve instances from various classes linked between them	36. nathan algren tom cruise Work	7.8
	37. rocky balboa sylvester stallone Work	11.9
	38. henry jaynes fonda work yours mine and ours character	3.5
	39. russell crowe work gladiator character	22.3
	40. brent spiner work star trek the next generation character	1.3

1540