

Automatically Creating Benchmarks for RDF Keyword Search Evaluation

Angelo B. Neves* · Luiz André P. Paes Leme ·
Yenier T. Izquierdo · Javier G. Jiménez · Giseli
R. Lopes · Marco A. Casanova

Received: date / Accepted: date

Abstract Keyword search systems provide users with a friendly alternative to access Resource Description Framework (RDF) datasets. Evaluating such systems requires adequate benchmarks, consisting of RDF datasets, keyword queries, and correct answers. However, available benchmarks often have small sets of queries and incomplete sets of answers, mainly because they are manually constructed with the help of experts. The central contribution of this article is an offline method to build benchmarks automatically, allowing larger sets of queries and more complete answers. The proposed method has two parts: query generation and answer generation. Query generation extracts keywords for each entity from a selected set of relevant entities, called inducers, and heuristics guide the process of extracting possible keywords related to each inducer. Answer generation takes the queries and computes solution generators (SG), which are subgraphs of the original dataset containing different answers to a query. Heuristics also guide the process by building SGs only for the relevant answers.

Angelo Batista Neves
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro/RJ, Brazil
E-mail: ajunior@inf.puc-rio.br

Luiz André P. Paes Leme
Universidade Federal Fluminense, Niterói/RJ, Brazil
E-mail: lapaesleme@ic.uff.br

Yenier Torres Izquierdo
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro/RJ, Brazil
E-mail: yizquierdo@inf.puc-rio.br

Javier Guillot Jiménez
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro/RJ, Brazil
E-mail: jguillot@inf.puc-rio.br

Giseli Rabello Lopes
Universidade Federal do Rio de Janeiro, Rio de Janeiro/RJ, Brazil
E-mail: giseli@ic.ufrj.br

Marco Antonio Casanova
Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro/RJ, Brazil
E-mail: casanova@inf.puc-rio.br

Keywords Benchmark · Keyword Search · RDF · heuristics · combinatorial issues

CR Subject Classification 10002951.10003317.10003359.10003360

1 Introduction

Keyword search is a popular information discovery method because it allows naive users to retrieve information without knowing schema details or query languages. 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. Recently, keyword search applications designed for Resource Description Framework (RDF) datasets (or RDF graphs) have emerged.

Tools for keyword search over RDF datasets, or *RDF-KwS tools*, have three main tasks: (1) retrieve nodes in the RDF graph that the keywords specify; (2) discover how they are interrelated to compose complete answers; and (3) rank these answers [19]. Hence, answers for keyword searches over RDF datasets are not just sets of nodes but sets of nodes and paths between them, i.e., subgraphs of the dataset.

RDF-KwS tools are evaluated using benchmarks with sets of information needs and their respective lists of correct answers, possibly ordered, for a given dataset. Despite the many existing benchmarks for structured data [6, 7, 9, 27], these benchmarks have at least four limitations when it comes to RDF-KwS: (1) they are frequently built for relational data; (2) they are incomplete in the sense that they do not cover many reasonable answers; (3) they are not always publicly available; (4) they are small.

To remedy the first limitation, some authors [11] adapted benchmarks developed for relational databases. However, the adaptation requires the triplication of relational databases and benchmark data, leading to problems when comparing different systems using different triplications.

As an example of the incompleteness of existing benchmarks, consider the keyword query “Mauritius India”, which is Query 43 for the Mondial dataset in Coffman’s benchmark¹. The list of correct answers in the benchmark covers just the organizations that both countries participate in. However, other answers that express a geographical relationship between Mauritius Islands and India should have been included in the list of correct answers. Incompleteness in this sense is a serious problem, which is difficult to overcome in manually constructed benchmarks.

Benchmarks size is also an issue. Many RDF-KwS systems are based on machine learning techniques and require a large volume of examples for training.

This article extends the paper “Automatic Construction of Benchmarks for RDF Keyword Search Systems Evaluation” (ICEIS’21) [3] by detailing the query generation process while addressing the general problem of constructing RDF keyword search benchmarks in a holistic approach, i.e., from the definition of keyword queries to the computation of correct answers. It is a challenging problem for three fundamental reasons: (1) RDF-KwS tools vary widely and compute differently – albeit correct – answers for the same keyword query; (2) computing the set of all correct answers

¹ Benchmark available at <https://dataverse.lib.virginia.edu/file.xhtml?fileId=1166&version=1.0>

for a given keyword query over an RDF dataset leads to an explosive combinatorial problem; and (3) the technique should be able to generate a large number of queries.

The main contributions of this article are as follows. First, the article introduces a method that, given an RDF dataset, automatically defines keyword queries and their respective correct answers. The method is designed as an offline process to benefit from less stringent response time constraints. It deals with the combinatorial nature of the problem by adopting heuristics based on *inducers*, *query generators*, *seeds*, and *solution generators*. Second, the article outlines an implementation of the method. Third, the article describes five benchmarks constructed with the proposed method and based on three real datasets, DBpedia, IMDb, and Mondial, and two synthetic datasets, LUBM and BSBM. Finally, the article compares the constructed benchmarks with keyword search benchmarks published in the literature.

The rest of this article is organized as follows. Section 2 covers related work. Section 3 contains the required definitions. Section 4 overviews the proposed method. Section 5 describes the automatic generation of keyword queries. Section 6 details the generation of answers. Section 7 evaluates the proposed benchmark generation method. Finally, Section 8 contains the conclusions and suggestions for future work.

2 Related Work

A crucial aspect of keyword search systems is their evaluation. In recent years, the research community concentrated on evaluating keyword search systems over relational databases [2] and entity retrieval [1]. Examples of relational benchmarks are Coffman’s benchmark [6], which uses Mondial, IMDb, and Wikipedia samples, and Oliveira’s benchmark [23], which uses Mondial, IMDb, DBLP, and Northwind.

Unfortunately, benchmarks to assess RDF keyword search systems are scarce [7]. To remedy this situation, some authors [11] adapted relational benchmarks to RDF. However, this approach depends on the triplification of relational databases and does not easily induce complete sets of correct query answers [15].

State-of-the-art RDF keyword search systems use different benchmarks, which are not always available, as shown in Table 1. For example, Dosso and Silvello [7] described openly available benchmarks over three real datasets, LinkedMDB, IMDb, and a subset of DBpedia [1], and two synthetic databases, the Lehigh University Benchmark (LUBM) [12] and the Berlin SPARQL Benchmark (BSBM) [5]. For IMDb, they defined 50 keyword queries and their correct translations to SPARQL queries. For DBpedia, the authors considered 50 topics from the classes `QALD2_te` and `QALD2_tr` of the *Question Answering over Linked Data (QALD)* campaigns². For the synthetic databases, they used 14 SPARQL queries for LUBM and 13 SPARQL queries for BSBM. For all original `SELECT` queries from these datasets, Dosso and Silvello mapped these queries to SPARQL `CONSTRUCT` queries and produced their equivalent keyword queries.

In particular, the Lehigh University Benchmark (LUBM) [12] is widely used to assess the performance of SPARQL engines, and was later extended [20] to cover full-text search performance.

² <http://qald.aksw.org>

Table 1: Summary of benchmarks used in state-of-the-art keyword search systems.

Tool	Ref.	Year	Description of Benchmark Used
SPARK	[31]*	2007	Database and keyword queries from Mooney Natural Language Learning Data
QUICK	[29]*	2009	An initial set of queries was extracted from a query log of the AOL search engine. Then, the queries were pruned based on the visited URLs, obtaining 3,000 sample keyword queries for IMDb and Lyrics Web pages. This process yielded 100 queries for IMDb, and 75 queries for Lyrics, consisting of 2–5 keywords.
	[26]*	2009	DBLP, TAP (http://tap.stanford.edu) and LUBM; 30 queries for DBLP, and 9 for TAP
	[6]†	2010	Samples of the Mondial, IMDb, and Wikipedia datasets; 50 queries for each dataset (not real user queries extracted from a search engine log).
	[10]*	2011	Datasets derived from the LibraryThing community and IMDb, and 15 queries for each dataset.
	[17]*	2014	Datasets: LUBM, Wordnet, BSBM, Barton and DBpedia Infobox. 12 Queries: 4 for LUBM, 2 for Wordnet, 2 for BSBM, 2 for Barton, 2 for DBpedia Infobox.
QUIOW	[30]	2016	DBpedia and Yago; queries derived from QALD-4
	[13]	2017	DBpedia+QALD-6 and Freebase* + Free917: an open QA benchmark that consists of NL question and answers pairs over Freebase.
KAT	[15]	2018	Full versions of the Mondial and IMDb datasets and queries from Coffman’s benchmark.
	[18]	2018	LUBM, Wordnet, BSBM, Barton, and DBpedia Infobox; four queries for LUBM, and queries for the other datasets.
QUIRA	[28]	2018	YAGO, DBLP and LUBM; nine queries for YAGO, three queries for DBLP, and six queries for LUBM.
	[25]	2018	AIFB and DBpedia; ten queries for each dataset (the sizes of the queries were between 2 and 8 keywords).
TSA+BM25 and TSA+VDP	[19]	2019	Full versions of IMDb and MusicBrainz; 50 queries from Coffman’s benchmark for IMDb, and 25 queries from QALD-2 for MusicBrainz. Details available at https://sites.google.com/view/quira/
TSA+BM25 and TSA+VDP	[7]	2020	Real datasets: LinkedMDB, IMDb, and a subset of DBpedia; 50 queries of Coffman’s benchmark for each dataset. Synthetic datasets: LUBM and BSBM; with 14 and 13 queries, respectively.

* Datasets have no public link or are not available for download.

† Benchmark for evaluating keyword search systems over relational databases.

However, none of these benchmarks was specifically designed for RDF keyword search.

The Question Answering over Linked Data challenges³ provide a series of queries to test Q&A approaches. Pound, Mika, and Zaragoza [24] addressed the task of ad-hoc object retrieval, as opposed to traditional ad-hoc document retrieval. They analyzed a real world Web query log from a semantic search point of view and proposed a semantic search query classification. Lastly, Balog and Neumayer [1] defined a test collection for entity search in DBpedia and summarized related work on entity search. Albeit attractive, in all these references, the proposed query workloads suffer from

³ <http://qald.aksw.org>

the same limitation - they retrieve individuals or lists of individuals - and do not fully explore the potentialities of RDF keyword search algorithms.

As for the keyword search itself, systems follow, basically, two distinct approaches. They can either take advantage of the declared schema of the dataset or disregard it and use the graph structure of the data to answer queries. QUIOW [15] (an improved implementation of previous work [11]) and QUICK [29] are both schema-based systems, but they differ in that QUICK relies on user feedback while QUIOW is fully automated. Both systems rewrite the original keyword query into SPARQL queries, with the help of the schema, to return the final answers.

SPARK [31] is a graph-based system that uses techniques, such as synonyms from WordNet and string metrics, to match keywords with RDF graph nodes. The matched nodes are then connected by minimum spanning trees from which SPARQL queries are generated. Elbassuoni and Blanco [10] described a technique to retrieve a set of subgraphs that match the keywords and to rank them based on statistical language models. Tran et al. [26] introduced the idea of generating summary graphs for the RDF graph to generate and rank candidate SPARQL queries. Lin et al. [18] summarized all the inter-entity relationships from the RDF data to translate keywords to SPARQL queries. Wen et al. [28] introduced a graph summarization technique that amounts to recovering an RDF schema from the RDF graph. Le et al. [17] also proposed to process keyword queries using an RDF graph summarization algorithm. Han et al. [13] described an algorithm that uses the keywords to first obtain elementary query graph building blocks and then applies a bipartite graph matching-based best-first search method to assemble the final query.

3 Definitions

An *RDF dataset* is a set \mathcal{T} of RDF triples (s, p, o) , which is equivalent to an edge-labeled directed graph $\mathcal{G}_{\mathcal{T}}$ (Figure 1a) such that the set of nodes of $\mathcal{G}_{\mathcal{T}}$ is the set of RDF terms that occur as subject or object of the triples in \mathcal{T} and there is an edge (s, o) in $\mathcal{G}_{\mathcal{T}}$, labeled p , iff the triple (s, p, o) occurs in \mathcal{T} .

An RDF dataset \mathcal{T} is also equivalent to a *knowledge base* $\mathcal{KB} = TBox \cup ABox$ in which a subset of triples $TBox \subseteq \mathcal{T}$ refers to terminological axioms and a subset $ABox = \mathcal{T} - TBox$ refers to assertional axioms, i.e., the $TBox$ defines the data schema, and the $ABox$ defines the data itself.

The *resource graph* induced by a subset $\mathcal{T}' \subseteq \mathcal{T}$ is the subgraph $\mathcal{R}\mathcal{G}_{\mathcal{T}'}$ of $\mathcal{G}_{\mathcal{T}'}$ obtained by dropping all literal nodes from $\mathcal{G}_{\mathcal{T}'}$. The nodes in $\mathcal{R}\mathcal{G}_{\mathcal{T}'}$ are the *resources* of \mathcal{T}' .

The *entity graph* induced by a subset $\mathcal{T}' \subseteq \mathcal{T}$ is the subgraph $\mathcal{E}\mathcal{G}_{\mathcal{T}'}$ of $\mathcal{G}_{\mathcal{T}'}$ obtained by dropping all literal nodes from the graph $\mathcal{G}_{\mathcal{T}' - TBox}$. The nodes in $\mathcal{E}\mathcal{G}_{\mathcal{T}'}$ are the *entities* of \mathcal{T}' .

The *neighborhood* of distance d of an entity e in \mathcal{T} , denoted ${}^d\mathcal{N}_{\mathcal{G}_{\mathcal{T}}}^e$, is the set of all nodes visited in a breadth-first walk of distance d starting from e in $\mathcal{E}\mathcal{G}_{\mathcal{T}}$.

The *schema graph* of \mathcal{T} , denoted $\mathcal{T}\mathcal{G}_{\mathcal{T}}$, is the graph in which the set of nodes are the named classes in $TBox$, and there is an edge (A, B) , labeled p in $\mathcal{T}\mathcal{G}_{\mathcal{T}}$, iff there is an edge labeled p from an entity of type A to an entity of type B in $ABox$, the

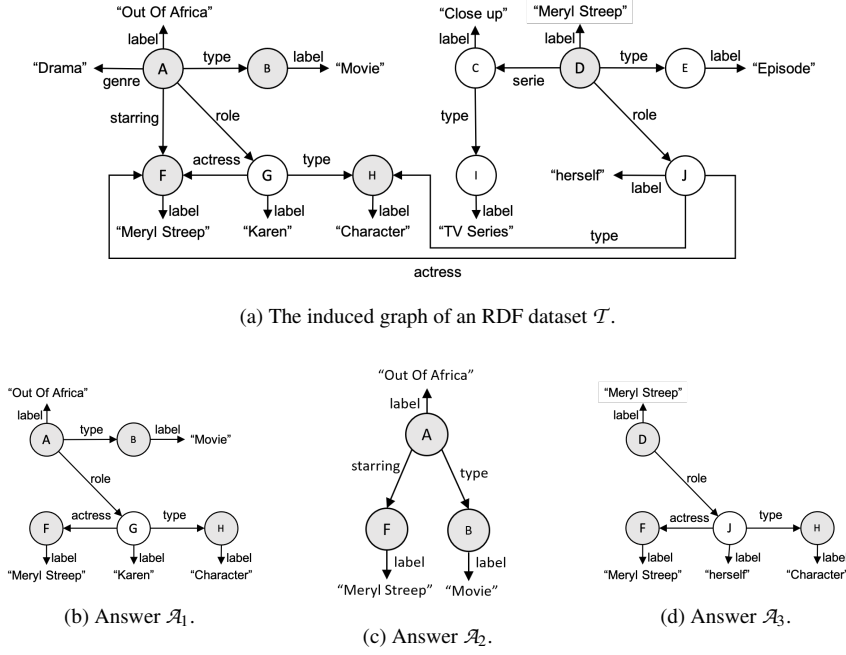


Fig. 1: Example of an RDF graph and three answers for the information need “*character of Meryl Streep in the movie Out of Africa*”.

domain of p contains a subset of A , i.e. $\neg(A \sqcap \exists p) \sqsubseteq \perp$, and the range of p contains a subset of B , i.e. $\neg(B \sqcap \exists p^-) \sqsubseteq \perp$, such that A and B are either asserted or inferred classes. The containment restrictions aim at deal with multiple class assertions. For example, let $\mathcal{KB} = \{A(s), B(o), C(s), p(s, o)\}$ be a knowledge base of a dataset \mathcal{T}''' . Without the containment restrictions one could add the edges (A, B) and (C, B) to $\mathcal{T}\mathcal{G}_{\mathcal{T}'''}$. However, if the $TBox$ defined restrictions on the domain and range of p , such as $\mathcal{KB} = \{A(s), B(o), C(s), p(s, o), \exists p \sqsubseteq A, \exists p^- \sqsubseteq B\}$, the only valid edge for $\mathcal{T}\mathcal{G}_{\mathcal{T}'''}$ would be (A, B) . The restrictions can also handle more complex domain and range definitions, such as $\exists p \sqsubseteq (A \sqcup C)$, $\exists p \sqsubseteq (A \sqcap C)$, and $\exists p \sqsubseteq (A \sqcup \exists q)$. Note that $\mathcal{T}\mathcal{G}_{\mathcal{T}}$ is not a replacement for the \mathcal{G}_{TBox} , it is just a convenient way of generating graph patterns for extracting keywords as shown later in Section 5.

A *schema cutout* of $\mathcal{T}\mathcal{G}_{\mathcal{T}}$, or simply a *cutout*, induced by ${}^d\mathcal{N}_{\mathcal{T}}^e$, denoted ${}^d\mathcal{C}\mathcal{G}_{\mathcal{T}}^e$, is the graph resulting from dropping nodes from $\mathcal{T}\mathcal{G}_{\mathcal{T}}$ that are not classes, either asserted or inferred, of any entity in ${}^d\mathcal{N}_{\mathcal{T}}^e$.

The *one-degree mirroring graph* of ${}^d\mathcal{C}\mathcal{G}_{\mathcal{T}}^e$, denoted ${}^{1,d}\mathcal{C}\mathcal{G}_{\mathcal{T}}$, is a graph that has a mirror node A^+ for each node A in ${}^d\mathcal{C}\mathcal{G}_{\mathcal{T}}^e$ and that if there is an edge (A, B) in ${}^d\mathcal{C}\mathcal{G}_{\mathcal{T}}^e$, there will be the edges (A, B) , (A^+, B^+) , (A^+, B) , and (A, B^+) in ${}^{1,d}\mathcal{C}\mathcal{G}_{\mathcal{T}}$. Figure 2 shows an example of a one-degree mirroring graph. Likewise, the *two-degree mirroring graph* ${}^{2,d}\mathcal{C}\mathcal{G}_{\mathcal{T}}$, will have nodes A , A^+ , and A^{++} and the edges (A, B) , (A^+, B^+) , (A^{++}, B^{++}) , (A, B^+) , (A, B^{++}) , and so on. An *n-degree mirroring graph* ${}^{n,d}\mathcal{C}\mathcal{G}_{\mathcal{T}}$ can thus be inductively defined, with ${}^{0,d}\mathcal{C}\mathcal{G}_{\mathcal{T}} = {}^d\mathcal{C}\mathcal{G}_{\mathcal{T}}^e$. The sets of mirrored

classes such as $\{A, A^+, A^{++}, \dots\}$ are called *mirrored sets*. The symbols A^+, A^{++}, \dots are distinct aliases for the same class A .

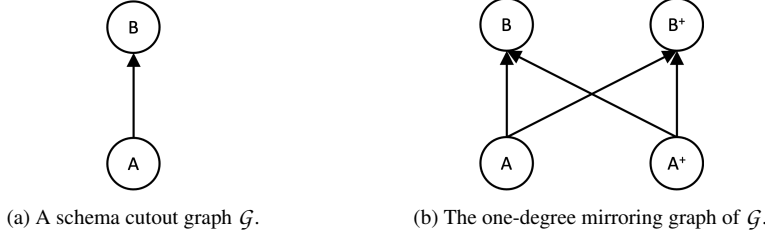


Fig. 2: Mirroring graph example.

A set of triples $\mathcal{T}' \subseteq \mathcal{T}$ induces a path $\pi = (s_0, p_0, s_1, p_1, s_2, \dots, s_n, p_n, s_{n+1})$ in $\mathcal{G}_{\mathcal{T}}$ iff, for each $i \in [0, n]$, either $(s_i, p_i, s_{i+1}) \in \mathcal{T}'$ or $(s_{i+1}, p_i, s_i) \in \mathcal{T}'$. We also say that (s_i, p_i, s_{i+1}) (or (s_{i+1}, p_i, s_i)) *occurs* in π . Note that we assume that paths in $\mathcal{G}_{\mathcal{T}}$ may traverse arcs in both directions. A path $\pi = (s_0, p_0, s_1, p_1, s_2, \dots, s_n, p_n, s_{n+1})$ *begins* (resp. *ends*) on a resource r iff $r = s_0$ or $r = p_0$ (resp. $r = s_{n+1}$ or $r = p_n$).

A *keyword query*, which represents an information need, is a set \mathcal{K} of literals.

A triple $(s, p, o) \in \mathcal{T}$ is a *matching triple* for \mathcal{K} iff o is a string literal that matches at least one keyword in \mathcal{K} . We also say that s is a *matching resource* for \mathcal{K} , the set \mathcal{M}_s of all matching triples in \mathcal{T} for \mathcal{K} whose subject is s is the set of *matching triples* of s in \mathcal{T} , and the graph induced by \mathcal{M}_s is the *matching subgraph* of s in $\mathcal{G}_{\mathcal{T}}$. Note that s may occur as the subject, property, or object of a triple in \mathcal{T} , but s is always the subject of a matching triple.

A set of triples $\mathcal{A} \subseteq \mathcal{T}$ is an *answer for \mathcal{K} over \mathcal{T}* iff \mathcal{A} can be partitioned into two subsets \mathcal{A}' and \mathcal{A}'' such that: (i) \mathcal{A}' is the set of all matching triples for \mathcal{K} in \mathcal{A} ; let $\mathcal{R}_{\mathcal{A}}$ be the set of subjects of such triples; (ii) $\mathcal{R}_{\mathcal{G}_{\mathcal{A}''}}$, the resource graph induced by \mathcal{A}'' , is connected and contains all resources in $\mathcal{R}_{\mathcal{A}}$. Condition (i) captures keyword matches and Condition (ii) indicates that an answer must connect the matching resources by paths in $\mathcal{R}_{\mathcal{G}_{\mathcal{A}''}}$. We also say that $\mathcal{R}_{\mathcal{A}}$ is the set of *matching resources* of \mathcal{A} and that $\mathcal{R}_{\mathcal{G}_{\mathcal{A}''}}$ is the *connectivity graph* of \mathcal{A} . Figure 1 shows an RDF graph and three answers for the keyword query $\mathcal{MS} = \{\text{“character”, “meryl”, “streep”, “movie”, “out”, “africa”}\}$.

This definition of answers is less stringent than those introduced by Bhalotia et al. [4], Hristidis and Papakonstantinou [14], Kimelfeld and Sagiv [16] since it neither requires all keywords to be matched nor includes a minimality criterion. For later reference, we define that an answer \mathcal{A} for \mathcal{K} over \mathcal{T} is *minimal* iff there is no proper subset \mathcal{B} of \mathcal{A} such that \mathcal{B} is an answer for \mathcal{K} and \mathcal{B} and \mathcal{A} have the same set of matching resources.

Finally, we can informally state the *RDF KWS-Problem* as: “Given an RDF dataset \mathcal{T} and a keyword query \mathcal{K} , find an answer \mathcal{A} for \mathcal{K} over \mathcal{T} , preferably, with as many keyword matches as possible and with the smallest set of triples as possible”.

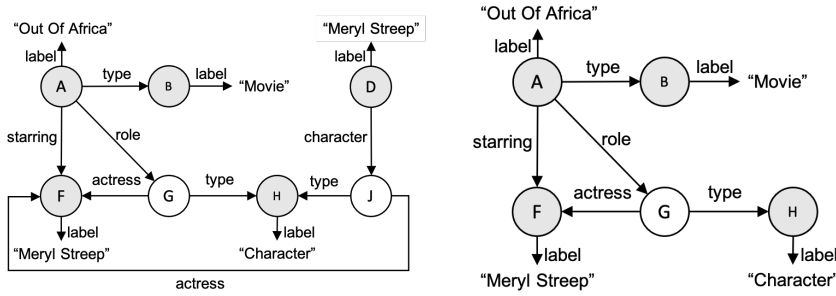
4 Overview of the Method for Computing Keyword Queries and Solution Generators

Let \mathcal{T} be an RDF dataset, and $\mathcal{EG}_{\mathcal{T}}$ be the entity graph induced by \mathcal{T} . This section outlines the proposed method for the automatic construction of sets of keyword queries over \mathcal{T} and ranked lists of correct answers for these queries.

The method starts by computing keyword queries, that is, sets of keywords \mathcal{K} , based on a set I of *inducing entities* or *inducers*, which are selected entities from \mathcal{T} (see Section 5 for details).

Let \mathcal{K} be a keyword query thus generated. Note that, although \mathcal{K} has been extracted for a particular inducer $i \in I$, there may exist subgraphs containing the keywords that have no relationship with i , but that, according to the definition, can also be considered correct answers to \mathcal{K} . The inducers then have the only purpose of deriving keyword queries connected in at least d $\mathcal{N}\mathcal{G}_{\mathcal{T}}^i$. This approach allows one to compute as many keyword queries are needed to create a benchmark.

The next step is to compute possible answers for \mathcal{K} . The method computes the complete set of matching resources for \mathcal{K} , called the *set of seeds* for \mathcal{K} , denoted $\mathcal{S}_{\mathcal{K}}$. By the previous definitions, the set of matching resources of an answer \mathcal{A} for \mathcal{K} must be a subset of $\mathcal{S}_{\mathcal{K}}$. Preferably, answers should contain subsets of $\mathcal{S}_{\mathcal{K}}$ that match all keywords in \mathcal{K} .



(a) The solution generator for $\mathcal{S}_{\mathcal{K}} = \{A, B, D, F, H\}$ from the dataset in Fig. 1a. (b) The solution generator for $\mathcal{S}_{\mathcal{K}} = \{A, B, F, H\}$ from the dataset in Fig. 1a.

Fig. 3: Examples of solution generators.

Consider the following example. Let “*character of Meryl Streep in the movie Out of Africa*” be an information need over the dataset \mathcal{T} of Figure 1.a. This information need can be translated to a keyword query $\mathcal{MS} = \{\text{“character”, “meryl”, “streep”, “movie”, “out”, “africa”}\}$. The set of seeds for \mathcal{MS} is $\mathcal{S}_{\mathcal{MS}} = \{A, B, D, F, H\}$. Figures 1.b–d show the graphs induced by three possible answers for \mathcal{MS} containing subsets of $\mathcal{S}_{\mathcal{MS}}$. Intuitively, answer \mathcal{A}_1 (Figure 1.b) is better than \mathcal{A}_2 (Figure 1.c) and \mathcal{A}_3 (Figure 1.d), because \mathcal{A}_1 addresses what seems to be the query intention, which is to find the character played by the actress in the movie. Also, \mathcal{A}_1 matches 6 of the 6 keywords in \mathcal{MS} , while \mathcal{A}_2 and \mathcal{A}_3 match only 5 and 3 keywords, respectively. Note that, in

\mathcal{A}_1 and \mathcal{A}_2 , but not in \mathcal{A}_3 , keywords are not matched by more than one literal. Also, note that the keywords “movie” and “character” are associated with elements of the schema, nodes B and H .

This example illustrates two important characteristics of keyword queries and correct answers: *keyword queries name existing resources, and answers correlate these resources*. In this example, “Meryl Streep”, “Out of Africa”, “character”, and “movie” are informal resource identifiers that represent an actress, a movie, the `Character` class, and the `Movie` class, respectively.

Let $\mathcal{R} \subseteq \mathcal{S}_{\mathcal{K}}$ be a subset of seeds. Assume that all seeds in \mathcal{R} belong to the same connected component of $\mathcal{R}\mathcal{G}_{\mathcal{T}}$, the resources graph induced by \mathcal{T} . One can compute all possible answers whose matching resources are subsets of \mathcal{R} by computing all acyclic paths in $\mathcal{R}\mathcal{G}_{\mathcal{T}}$ between pairs of distinct nodes in \mathcal{R} , combining them in all distinct ways to construct connected graphs containing all nodes in \mathcal{R} , and adding the matching subgraphs for the seeds $r \in \mathcal{R}$.

Consider the set of seeds $\mathcal{R}'_{\mathcal{M}\mathcal{S}} = \{A, B, F, H\}$, for example. If one selects the paths $(B \leftarrow A \rightarrow G \rightarrow F)$ and $(G \rightarrow H)$ and adds the matching subgraphs $(A \rightarrow \text{“Out of Africa”})$, $(B \rightarrow \text{“Movie”})$, $(F \rightarrow \text{“Meryl Streep”})$, and $(H \rightarrow \text{“Character”})$, one will obtain the answer in Figure 1.b. If one selects the path $(B \leftarrow A \rightarrow F \leftarrow G \rightarrow H)$ and adds the same matching subgraphs, one will obtain another valid answer (not shown in the figure).

Consider now the set of seeds $\mathcal{R}''_{\mathcal{M}\mathcal{S}} = \{A, B, F\}$. If one selects the path $(B \leftarrow A \rightarrow F)$ and adds the matching subgraphs $(A \rightarrow \text{“Out of Africa”})$, $(B \rightarrow \text{“Movie”})$, and $(F \rightarrow \text{“Meryl Streep”})$, one will obtain the answer in Figure 1.c. If one selects a different path $(B \leftarrow A \rightarrow G \rightarrow F)$, one will obtain yet another answer. In general, distinct path combinations may lead to distinct valid answers for the same set of seeds.

More precisely, let \mathcal{K} again be a keyword query and $\mathcal{R} \subseteq \mathcal{S}_{\mathcal{K}}$. Assume that all seeds in \mathcal{R} belong to the same connected component of $\mathcal{R}\mathcal{G}_{\mathcal{T}}$. A set of triples $\mathcal{S}\mathcal{G} \subseteq \mathcal{T}$ is a *solution generator* for \mathcal{R} iff $\mathcal{S}\mathcal{G}$ can be partitioned into two sets, $\mathcal{S}\mathcal{G}'$ and $\mathcal{S}\mathcal{G}''$, such that: (i) $\mathcal{S}\mathcal{G}'$ is the set of all matching triples of the resources in \mathcal{R} ; (ii) $\mathcal{S}\mathcal{G}''$ is the set of all triples that occur in paths in $\mathcal{R}\mathcal{G}_{\mathcal{T}}$ that begin and end on the seeds in \mathcal{R} . We say that $\mathcal{S}\mathcal{G}$ *expresses* an answer \mathcal{A} iff $\mathcal{A} \subseteq \mathcal{S}\mathcal{G}$ and the set of matching resources of \mathcal{A} is \mathcal{R} .

Figure 3 shows the solution generators for $\mathcal{R}_{\mathcal{M}\mathcal{S}} = \{A, B, D, F, H\}$ and $\mathcal{R}'_{\mathcal{M}\mathcal{S}} = \{A, B, F, H\}$. Note that, even though both $\mathcal{R}_{\mathcal{M}\mathcal{S}}$ and $\mathcal{R}'_{\mathcal{M}\mathcal{S}}$ cover all keywords, they are distinct and express distinct sets of answers.

A *synthetic benchmark* is then a triple $s = (\mathcal{T}, Q_s, A_s)$ such that \mathcal{T} is an RDF dataset, Q_s is a list of keyword queries, and A_s contains, for each query in Q_s , a list of solution generators. Section 7 illustrates this concept.

However, computing all solution generators can be expensive, depending on the cardinality of $\mathcal{S}_{\mathcal{K}}$ and the number of paths between the seeds. We then propose to compute solution generators that capture only the most relevant answers. Fortunately, unlike traditional Information Retrieval (IR) systems, one can exploit peculiarities of the RDF KwS-Problem to define optimization heuristics that reduce the computational cost and help rank solution generators.

The differences between traditional IR systems and RDF-KwS systems stem from the fact that keywords that co-occur in a document may not convey the idea of keyword correlation. By contrast, an RDF subgraph connecting resources linked to these keywords is much more likely to be related to the intended meaning of the keyword query because resources are not connected by chance in an RDF graph, as it may be the case in a text document.

By assuming such differences, one can define some characteristics of good answers: the keywords must be connected as closely as possible, and answers must contain as many keywords as possible. One can define other features based on the number of resources in answers and the co-occurrence of keywords among the literals. These features may guide the automatic computation of answers by helping prune the search space.

Section 6 discusses four heuristics to work around the complexity of the problem of computing solution generators, guided by five questions: (1) Are all seeds relevant?; (2) Are all paths between seeds relevant?; (3) Should we prefer answers that match more literals or answers that match fewer literals?; (4) Should we prefer answers in which literals in the keyword query occur in many seeds, or answers in which literals occur in only one seed?; (5) Should we prefer answers with many seeds or answers with fewer seeds?

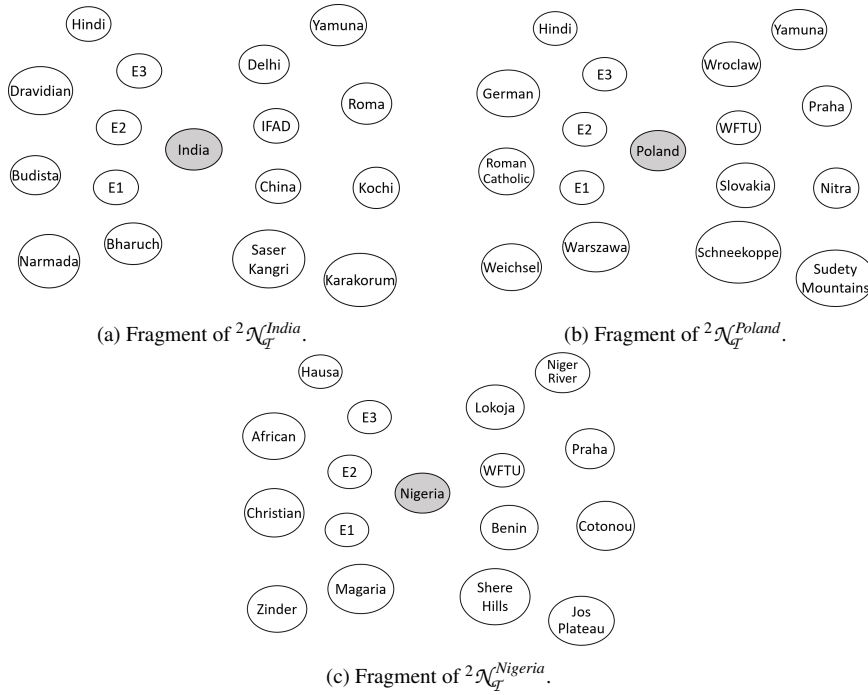


Fig. 4: Neighborhood fragments.

5 Computing Keyword Queries

This section describes, with the help of examples, how to automatically generate sets of keyword queries for a given RDF dataset \mathcal{T} .

An *inducer function* for \mathcal{T} is a function that maps \mathcal{T} into a set I of entities of \mathcal{T} , called *inducers*. Such function should follow requirements consistent with the benchmark's purpose, i.e., it should select entities from the information domain in question with appropriate relevance scores. High scores induce keywords from relevant entities, while low scores can challenge RDF-KwS systems by inducing keywords from less relevant entities. For example, let \mathcal{T} be the Mondial RDF dataset⁴. An inducer function for \mathcal{T} would select the *top-k and bottom-k* countries according to their infoRank score [19]. InfoRank estimates entities' relevance for users based on three intuitions: 1) important things have lots of datatype properties, 2) important things are surrounded by other important things, and 3) few good friends are better than many acquaintances. The *first step* of the process for generating queries then consists in selecting a set of inducers.

The *second step* is to compute the inducers' neighborhoods ${}^d\mathcal{N}_{\mathcal{T}}^i$, $i \in I$. Figure 4a shows an example fragment of the neighborhood ${}^2\mathcal{N}_{\mathcal{T}}^{India}$ of the country *India*, which is the 8th country with the most significant infoRank score in the Mondial dataset.

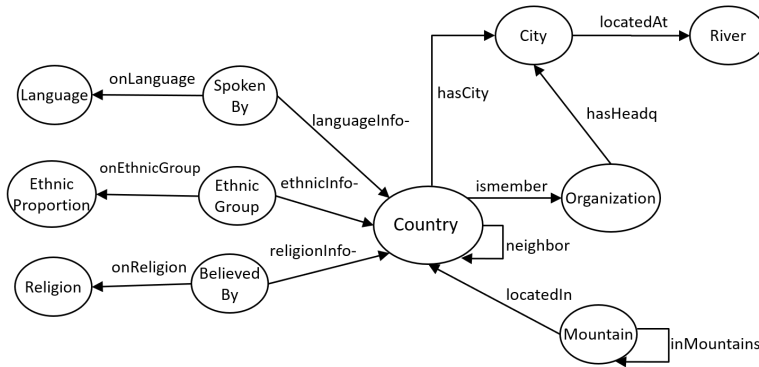


Fig. 5: A fragment of the cutout schema ${}^2\mathcal{C}\mathcal{G}_{\mathcal{T}}^{India}$ induced by the neighborhood graph of India, which is coincidentally similar to ${}^2\mathcal{C}\mathcal{G}_{\mathcal{T}}^{Poland}$ and ${}^2\mathcal{C}\mathcal{G}_{\mathcal{T}}^{Nigeria}$.

The *third step* is to compute the n -degree mirroring graphs ${}^{n,d}\mathcal{C}\mathcal{G}_{\mathcal{T}}^i$, and the set of all possible minimal Steiner trees in each one. Figure 5 shows a fragment of the cutout schema ${}^2\mathcal{C}\mathcal{G}_{\mathcal{T}}^{India}$ which is coincidentally similar to ${}^2\mathcal{C}\mathcal{G}_{\mathcal{T}}^{Poland}$ and ${}^2\mathcal{C}\mathcal{G}_{\mathcal{T}}^{Nigeria}$. The Steiner trees induce different SPARQL graph patterns P_i that correlate entities from ${}^d\mathcal{N}_{\mathcal{T}}^i$. Note that n is a user-defined parameter that allows one to control the cardinality of elements of the same type in the graph patterns.

⁴ <http://www.dbis.informatik.uni-goettingen.de/Mondial/>

These graph patterns, called *answer graph patterns*, represent typical answer templates for keyword sets. For example, Figure 6a shows a Steiner tree from ${}^{1,2}CG_{\mathcal{T}}^{India}$ that induces the graph pattern in Appendix A.1. Note that the expression *typical answer template* refers to the process that guides the query generation. It does not mean that it is the most common or desired answer pattern among all valid answers. There can be many more valid answer graph patterns that search systems can find for the same keyword set, as explained in Section 6.

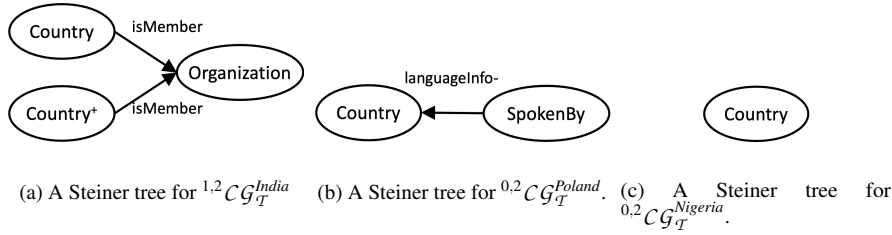


Fig. 6: Example of Steiner trees.

Each node of a Steiner tree maps to distinct SPARQL variables, the edges of the trees straightforwardly derive the joining triple patterns, and the types of the nodes filter out entities of their respective types. The bind expression determines that solutions must contain the inducer and the filter dismiss solutions with repetitive entities. This last restriction aims to dismiss solutions such as $?s1=:IND$ and $?s2=:IND$, where $:IND$ is the India’s URI. It seems not to make sense in this pattern. We opted to automatically generate the full mutual inequality filter rather than a more specific one such as `FILTER (?s1!=?s2)` to avoid more sophisticated inferences, although this strategy can dismiss good repetitions. For the sake of query generation, it is not required to generate all possible ones.

Keywords are then extracted for each entity, property, and class in the patterns by instantiating P_i from \mathcal{T} and retrieving values of their datatype properties. The query in Appendix A.2 shows the value extraction for the pattern in Appendix A.1, and Table 2 shows fragments of two binding sets from the Mondial dataset. We assume that all terminological elements have a single property `rdfs:label`. More complex queries could avoid such assumptions, which is disregarded here for the sake of conciseness. Lines 12–14 extract label values for schema elements in the pattern. Lines 16–27 extract label values from all properties for each entity in the pattern. Lines 29–38 extract literal values from all properties for each entity in the pattern.

?v1	?v2	?v3	...	?v1	?v2	?v3
“Country”	“Organization”	“is member”	...	“India...”	“Mauritius...”	“IFAD”
“Country”	“Organization”	“is member”	...	“India...”	“Mauritius...”	“G-77”

Table 2: A fragment of the binding sets for the query in Appendix A.2.

Each line of Table 2 corresponds to a different entity instantiation set and, therefore, it may be considered a different answer for the pattern. Still, each line may produce distinct keyword sets.

Table 2 enables benchmark developers to create queries, such as queries number 43, 30, and 27 of Coffman’s benchmark for Mondial, $\mathcal{K}_{\text{Coff-Q43}} = \{\text{“mauritius”}, \text{“india”}\}$, $\mathcal{K}_{\text{Coff-Q30}} = \{\text{“poland”}, \text{“language”}\}$, and $\mathcal{K}_{\text{Coff-Q27}} = \{\text{“nigeria”}, \text{“gdp”}\}$, respectively. The first query contains keywords for the names of two countries, the second query contains keywords for the name of a country and the name of an object property, and the third contains keywords for the name of a country and one of its datatype properties (Gross Domestic Product - GDP).

The *fourth and final step* of the keyword-query generation is to extract keywords from the bindings. For example, the 43rd Coffman’s query, $\mathcal{K}_{\text{Coff-Q43}} = \{\text{“mauritius”}, \text{“india”}\}$, can be created by extracting the literals from the bindings for $?v_1$ and $?v_2$. The SPARQL query in Appendix A.3 can generate Coffman’s ground truth for this query. Each created named graph is considered a different solution. This ground truth is adapted from Coffman’s benchmark and our previous answer definition for RDF-KwS since Coffman’s benchmark is built for relational databases and does not define answers as graphs.

The query in Appendix A.3 is not necessarily the query an RDF-KwS system would generate to answer $\mathcal{K}_{\text{Coff-Q43}}$. It simply shows an intuition that the method can generate queries similar to Coffman’s associated with the same ground truth. Section 6 defines the actual ground truths.

Each tuple of bindings in Table 2 can derive many more keyword queries, each literal combination would be a different query. However, some combinations are not allowed: those with no literals coming from property values or classes of any leaf entity. *Leaf entities* are instances of leaf classes. *Leaf classes* are those connected by just one edge in Steiner trees. The leaf entities for the Steiner tree in Figure 6a are bound to the variables $?s_1$ and $?s_2$ in Appendix A.1. They are instances of the class *Country*. Recall that *Country*⁺ is an alias to the class *Country*. We call this restriction the *must-include-all-leaves rule*. Such restriction is necessary because keyword search systems usually try to interconnect nodes linked to the given keywords. For example, Coffman’s 43rd query intends to connect the nodes corresponding to India and Mauritius. Search systems would try to find paths to connect these nodes. Eventually, they would find IFAD and G-77, representing international organizations in which both countries participate. If, on the other hand, search systems used the keywords $\{\text{India}, \text{IFAD}\}$, the algorithms could never find Mauritius since a path interconnecting India and IFAD traversing Mauritius could not exist. Those keywords then could not be appropriate for the pattern in Figure 6a.

Because of the combinatorial nature of this problem, we propose not to materialize every keyword combination. Instead, we extract keywords from every binding for each binding set (each line of Table 2) and leave the generation of appropriate keyword combinations to benchmark developers. The method outputs for each inducer a set of pairs (K, G) , called *keyword query generators*, such that K is a set of bindings, in the form of Table 2, and G is an answer graph pattern. The pairs (K, G) allow developers to follow the *must-include-all-leaves rule* to generate their own keyword sets.

Query generators are also convenient to compare different benchmarks in terms of the variety of expected answers. Sets of expected answers that show considerable variety pose more challenges to search engines. For example, from the 50 queries provided by Coffman for the Mondial database, one can count just 14 answer patterns. Indeed, for instance, Queries 36–45 aim at discovering relationships between two countries and international organizations and, therefore, have the same answer pattern. Differently from the existing benchmarks, the proposed method can generate many distinct answer patterns.

A second example illustrates a limitation of the method and how it can generate a query similar to Query 30 of Coffman’s benchmark for Mondial: $\mathcal{K}_{\text{Coff-Q30}} = \{\text{“poland”}, \text{“language”}\}$. This query aims at finding the language spoken in Poland and could be generated with the Steiner tree in Figure 5b. Poland is the name of an instance of type `:Country` and language is the label of the property `:onLanguage` with domain `:SpokenBy` which points to the Language entity `:Polish`. However, the must-include-all-leaves rule requires one to extract keywords from property values of `:SpokenBy` instances or from the class itself. If one replaced the original query with $\{\text{“poland”}, \text{“spoken”}, \text{“language”}\}$ the answer graph pattern would be that in Appendix B.1, the extraction query that in Appendix B.2, the binding table that in Table 3, and the ground truth that in Appendix B.3. The keywords would be extracted from $?12$, $?15$, and $?v1$.

$?11$	$?12$...	$?15$	$?v1$...
“Country”	“Spoken By”	...	“... on language...”	“Poland...”	...

Table 3: A fragment of the binding sets for the query in Appendix B.2.

The last example generates Query 27 of Coffman’s benchmark for Mondial: $\mathcal{K}_{\text{Coff-Q27}} = \{\text{“nigeria”}, \text{“gdp”}\}$. Nigeria is the 15th country by ranking with the infoRank score, and *gdp* is the Gross Domestic Product datatype property. The Steiner tree is in Figure 6c, the answer patterns are in Appendix C.1, the extraction query is in Appendix C.2, the binding table is Table 4, and the ground truth is in Appendix C.3. The keywords would be extracted from $?12$, and $?v1$.

$?11$	$?12$	$?v1$
“Country”	“... gdp...”	“Nigeria”

Table 4: A fragment of the binding sets for the query in Appendix C.2.

Computing all Steiner trees for ${}^{n,d}\mathcal{CG}_T^i$ can be computationally expensive. Besides limiting n and d , we then defined the following heuristics to filter out unwanted patterns: 1) dismiss Steiner trees with more than ρ_1 nodes; 2) dismiss Steiner trees that do not include the inducer’s classes; 3) dismiss Steiner trees with a max distance

between two nodes longer than ρ_2 ; and 4) dismiss Steiner trees with more than ρ_3 mirrored sets.

The first heuristic expresses the intuition that small solutions are preferable to complex ones. The complexity is estimated by the number of nodes since minimal Steiner trees, by definition, contain the smallest possible set of edges for a given set of terminal nodes. The second heuristic expresses the intuition that inducers guide the query generation process. The third heuristic expresses the intuition that users do not easily understand distant relationships, as argued by Nunes et al. [21]. Finally, the last heuristic allows one to control the answer patterns. For example, solutions with two countries and two organizations in an answer graph pattern may seem less satisfactory than the graph pattern in Figure 6a. The third parameter (ρ_3) then limits the number of mirrored sets in the patterns.

6 Computing Solution Generators

This section addresses the computation of solution generators for keyword queries through four heuristics to circumvent the complexity of the problem.

The first heuristic refers to the selection of the most relevant seeds. Assume that Lucene for Apache Jena Fuseki is the text search engine over RDF adopted. The *Lucene score* is a TF-IDF-based score that captures the relevance of property values for the keywords. The complete set of seeds of a keyword query can be obtained from the Lucene inverted index, which can be a large set.

One could then limit this set to the top- k resources by the Lucene score, but the resource labeled “Meryl Streep” would be the 7th entry in the ranked list of seeds, and the resource labeled “Out of Africa” would be the 30th entry. If one took the top 30 resources in the ranking, the two seeds would be selected, but many less relevant seeds would also be selected.

Let \mathcal{K} be keyword query and $S_{\mathcal{K}}$ be the set of seeds of \mathcal{K} . We define the *entity score* of a seed $s \in S_{\mathcal{K}}$ for \mathcal{K} as

$$es(s, \mathcal{K}) = \frac{1}{2}(\max_{v_j}(lucene(s, v_j, \mathcal{K})) + infoRank(s)) \quad (1)$$

where v_j is a string value such that there is a triple $(s, p, v_j) \in \mathcal{T}$. The infoRank score [19] reflects the relevance of s to users. The entity score ranges in $[0, 1]$, since we used normalized versions of $lucene(s, v_j, \mathcal{K})$ and $infoRank(s)$. If a text search engine other than Lucene is adopted or a resource relevance measure other than infoRank is used, Eq. 1 should be adjusted accordingly.

By ranking the set of seeds of \mathcal{K} according to the entity score, the resource labeled “Meryl Streep” would appear in the 1st position, and that labeled with “Out of Africa” would appear in the 18th position.

The *first heuristic* is then to refine the set of seeds $S_{\mathcal{K}}$ to be the set $\Sigma_{\mathcal{K}}$ of the top- σ_2 elements of $\{s \in S_{\mathcal{K}} | es(s, \mathcal{K}) \geq \sigma_1\}$ ordered in decreasing order of entity score, where σ_1 and σ_2 are thresholds empirically defined to optimize computing resources and match user’s preferences.

The threshold σ_1 defines a minimum seed entity score to speed up the Lucene engine for practical reasons, and σ_2 effectively limits the number of selected seeds. By defining $\sigma_1 = 0$ and $\sigma_2 = \infty$, one would select the full set of seeds. But, by defining $\sigma_1 > 0$ and $\sigma_2 < \infty$ one can restrict the cardinality of $2^{\Sigma_{\mathcal{K}}}$ and, consequently, the total number of paths to compute.

However, $\Sigma_{\mathcal{K}}$ may not cover all keywords in \mathcal{K} . We then compute another set $\mathcal{S}_{\mathcal{K}_i}$ of matching resources, where \mathcal{K}_i is the subset of \mathcal{K} not matched by resources in $\Sigma_{\mathcal{K}}$, and repeat this procedure to extend $\Sigma_{\mathcal{K}}$ until no other keyword can be matched or $\mathcal{K}_i = \{\}$. More precisely, for any $\mathcal{K}_i \subseteq \mathcal{K}$, let $\mathcal{S}_{\mathcal{K}_i}$ be the set of seeds in $\mathcal{S}_{\mathcal{K}}$ that match keywords in \mathcal{K}_i . Let $\mu[\sigma_1](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i}) = \{s \in \mathcal{S}_{\mathcal{K}_i} | es(s, \mathcal{K}_i) \geq \sigma_1\}$, and $\tau[\sigma_1, \sigma_2](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i})$ be the top- σ_2 elements of $\mu[\sigma_1](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i})$, ordered by the entity score of the seeds.

Let $(\mathcal{K}_0, \dots, \mathcal{K}_N)$ be the longest sequence such that $\mathcal{K}_0 = \mathcal{K}$ and, for each $i \in [1, N]$, \mathcal{K}_i is the set of keywords in \mathcal{K} not matched by seeds in $\tau[\sigma_1, \sigma_2](\mathcal{K}_{i-1}, \mathcal{S}_{\mathcal{K}_{i-1}})$ and $\mathcal{K}_i \neq \emptyset$ and $\mathcal{S}_{\mathcal{K}_i} \neq \emptyset$. Then, $\Sigma_{\mathcal{K}}$ is defined as

$$\Sigma_{\mathcal{K}} = \bigcup_{i=0}^N \tau[\sigma_1, \sigma_2](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i}) \quad (2)$$

The *second heuristic* refers to the selection of sets in $2^{\Sigma_{\mathcal{K}}}$ for which one would compute the solution generators. It is done by scoring each $\mathcal{R} \in 2^{\Sigma_{\mathcal{K}}}$ and selecting the top-ranked ones based on four principles. First, the set of matching keywords of \mathcal{R} is the union of the set of keywords matched by each resource in \mathcal{R} ; the sets \mathcal{R} with the most significant number of keyword matchings are preferable and potentially would generate better answers. Second, the keywords should preferably match just a few seeds of an answer because keywords identify entities. We assume that answers where keywords do not match more than one resource, such as those in Figures 1.b and 1.c, are more relevant.

Nevertheless, as detailed later in this section, this constraint can be relaxed to allow answers such as that in Figure 1d. Third, smaller answers, in terms of the number of seeds, are preferable over larger ones since they are easier to understand. Lastly, small sets are preferable, but it is also necessary that their resources are the most relevant to users. Based on these principles, we define the following scores.

Let \mathcal{E} be a set of resources. The *coverage score* of \mathcal{E} , denoted $cs(\mathcal{E}, \mathcal{K})$, measures the number of keywords in \mathcal{K} matched by resources in \mathcal{E} ; $cs(\mathcal{E}, \mathcal{K}) = 1$, if all keywords are matched, and $cs(\mathcal{E}, \mathcal{K}) = 0$, if no keyword is matched:

$$cs(\mathcal{E}, \mathcal{K}) = \frac{\sum_{k_i \in \mathcal{K}} occur(\mathcal{E}, k_i)}{|\mathcal{K}|} \quad (3)$$

where $occur(\mathcal{E}, k_i) = 1$, if some resource in \mathcal{E} matches k_i , and $occur(\mathcal{E}, k_i) = 0$, otherwise.

The *co-occurrence score* of \mathcal{E} , denoted $os(\mathcal{E}, \mathcal{K})$, measures the repetition degree of keywords in \mathcal{K} among resources in \mathcal{E} ; $os(\mathcal{E}, \mathcal{K}) = 1$, if each keyword is matched by only one resource, and $os(\mathcal{E}) = 0$, if all keywords are matched by all resources:

$$os(\mathcal{E}, \mathcal{K}) = \frac{1 - \frac{f(\mathcal{E}, \mathcal{K})}{|\mathcal{E}|}}{1 - \frac{1}{|\mathcal{E}|}} \quad (4)$$

where $f(\mathcal{E}, \mathcal{K})$ is the average number of resources in \mathcal{E} that match keywords in \mathcal{K} . Keywords not covered in \mathcal{E} are not taken into account; $os(\mathcal{E}, \mathcal{K})$ is assumed to be 1, if the denominator is 0.

Let C be the collection of sets of resources considered. The *size score* of \mathcal{E} w.r.t. C , denoted $ss(\mathcal{E})$, measures the relative size of \mathcal{E} ; $ss(\mathcal{E}) = 1$, if \mathcal{E} is one of the smallest sets, and $ss(\mathcal{E}) = 0$, if \mathcal{E} is one of the largest sets:

$$ss(\mathcal{E}) = \frac{\mathcal{N} - |\mathcal{E}|}{\mathcal{N} - 1} \quad (5)$$

where \mathcal{N} is the cardinality of the largest set in C ; $ss(\mathcal{E}, \mathcal{K})$ is assumed to be 1, if the denominator is 0.

The *infoRank score* of \mathcal{E} , denoted $is(\mathcal{E})$, is the average infoRank value [19] of the resources in \mathcal{E} :

$$is(\mathcal{E}) = average(\{infoRank(s_i) | s_i \in \mathcal{E}\}) \quad (6)$$

The second heuristic is then the refinement of the set $2^{\Sigma_{\mathcal{K}}}$ by choosing only those $\mathcal{R} \in 2^{\Sigma_{\mathcal{K}}}$ with better coverage, lower co-occurrence, fewer number of resources, and with more relevant nodes. Recall that a lower co-occurrence would favor answers such as those in Figures 1.b and 1.c, while allowing answers such as that in Figure 1.d. On the other hand, no co-occurrence ($os(\mathcal{R}, \mathcal{K}) = 1$) would allow answers such as those in Figures 1.b and 1.c only. The refinement is expressed by defining set Π_K as follows:

$$\Pi_K = \{\mathcal{R} \in 2^{\Sigma_{\mathcal{K}}} | cs(\mathcal{R}, \mathcal{K}) \geq \sigma_3 \wedge os(\mathcal{R}, \mathcal{K}) \geq \sigma_4 \wedge ss(\mathcal{R}) \geq \sigma_5 \wedge is(\mathcal{R}) \geq \sigma_6\} \quad (7)$$

where σ_3 , σ_4 , σ_5 , and σ_6 are empirically defined according to the available computing resources and user preferences. If $\sigma_3 = \sigma_4 = \sigma_5 = \sigma_6 = 0$ then $\Pi_K = 2^{\Sigma_{\mathcal{K}}}$ and all possible solution generators with $\Sigma_{\mathcal{K}}$ would be computed. The above scores can be redefined for subsets of triples $\mathcal{U} \subseteq \mathcal{T}$ by taking $\mathcal{E} = \mathcal{E}_{\mathcal{U}}$, where $\mathcal{E}_{\mathcal{U}}$ is the set of all resources in \mathcal{U} .

The *third heuristic* is to consider only paths between seeds with length less than or equal to a given limit L , say $L = 4$, to compute solution generators. As argued by Nunes et al. [21], paths longer than four would express unusual relationships, which users might misinterpret.

The *fourth heuristic* is to rank the solution generators in Π_K according to their scores, following user needs. For example, if one ranks based only on the coverage and size, one may define a Boolean function “order” between solution generators.

$$order(\mathcal{S}\mathcal{G}_1, \mathcal{S}\mathcal{G}_2) = \begin{cases} cs(\mathcal{S}\mathcal{G}_1, \mathcal{K}) \geq cs(\mathcal{S}\mathcal{G}_2, \mathcal{K}), & \text{if } C \text{ holds} \\ ss(\mathcal{S}\mathcal{G}_1, \mathcal{K}) \geq ss(\mathcal{S}\mathcal{G}_2, \mathcal{K}), & \text{otherwise} \end{cases} \quad (8)$$

where C is the condition $cs(\mathcal{S}\mathcal{G}_1, \mathcal{K}) \neq cs(\mathcal{S}\mathcal{G}_2, \mathcal{K})$. Alternatively, one could rank solution generators by their average scores as in Eq. 9 to balance the losses and gains of each individual score:

$$order(\mathcal{S}\mathcal{G}_1, \mathcal{S}\mathcal{G}_2) = \frac{1}{4}(cs(\mathcal{S}\mathcal{G}_2, \mathcal{K}) + os(\mathcal{S}\mathcal{G}_2, \mathcal{K}) + ss(\mathcal{S}\mathcal{G}_2) + is(\mathcal{S}\mathcal{G}_2)) \quad (9)$$

Eqs. 2, 7, 8, and 9 are, in fact, flexibilization points of the method. For example, instead of using the Lucene score in Eq. 1, one could use a keyword count, and instead of a selection in Eq. 7, one could use the top of the ranking of the sets $\mathcal{R} \in 2^{\Sigma_{MS}}$ with the order function defined in Eq. 9. The method chosen will determine the set of solution generators that will be computed. The central contribution of this work lies, then, in using the peculiarities of RDF keyword search to define a process for optimizing the computation of solution generators.

Algorithm 1 Algorithm to compute a ranked list of solution generators for a keyword query \mathcal{K} over an RDF dataset \mathcal{T} .

Require: a keyword query \mathcal{K} and an RDF dataset \mathcal{T}

```

1:  $\Sigma_{\mathcal{K}} = \bigcup_{i=0}^N \tau[\sigma_1, \sigma_2](\mathcal{K}_i, \mathcal{S}_{\mathcal{K}_i})$ 
2:  $\Pi_K = \{\mathcal{R} \in 2^{\Sigma_{\mathcal{K}}} \mid cs(\mathcal{R}, \mathcal{K}) \geq \sigma_3 \wedge os(\mathcal{R}, \mathcal{K}) \geq \sigma_4 \wedge ss(\mathcal{R}) \geq \sigma_5 \wedge is(\mathcal{R}, \mathcal{K}) \geq \sigma_6\}$ 
3:  $\mathcal{U} = \{\}$ 
4: for all  $\mathcal{R} \in \Pi_K$  do
5:    $\mathcal{S}\mathcal{G} = \{\}$ 
6:   for all distinct unordered pairs of nodes  $\{n_1, n_2\}$  such that  $n_1, n_2 \in \Sigma_{\mathcal{K}}$  do
7:      $\mathcal{S}\mathcal{G} = \mathcal{S}\mathcal{G} \cup \{(s, p, o) \in \mathcal{T} \mid (s, p, o) \text{ is in a path with length } \leq 4 \text{ between } n_1 \text{ and } n_2\}$ 
8:   end for
9:   if  $G'_{\mathcal{S}\mathcal{G}}$  is connected then
10:     $\mathcal{U} = \mathcal{U} \cup \{\mathcal{S}\mathcal{G}\}$ 
11:   end if
12: end for
13: Create  $\mathcal{U}'$  by ordering  $\mathcal{U}$  using the score function in Eq. 8
14: return  $\mathcal{U}'$ 

```

Algorithm 1 embeds the proposed heuristics to reduce the cost of computing solution generators by disregarding the less relevant ones. It takes as input a keyword query \mathcal{K} and an RDF dataset \mathcal{T} , and outputs a ranked list of solution generators according to the score function in Eq. 8. Lines 1 and 2 prepare the sets of nodes that will guide the computation of the solution generators according to Eqs. 2 and 7. Lines 4–12 compute solution generators for each set of seeds in Π_K . In lines 9–11, if the set of triples $\mathcal{S}\mathcal{G}$ computed for a set of seeds $\mathcal{R} \in \Pi_K$ does not induce a connected graph $G'_{\mathcal{S}\mathcal{G}}$ (connectivity graph of $\mathcal{S}\mathcal{G}$), then $\mathcal{S}\mathcal{G}$ is discarded because, for each connected component C_i of $G'_{\mathcal{S}\mathcal{G}}$, there is a strict subset \mathcal{R}_j of \mathcal{R} such that the set of triples computed for \mathcal{R}_j induces C_i .

7 Evaluation of the Benchmark Generation Method

This section addresses the critical question of evaluating the proposed benchmark generation method. The evaluation concentrates on assessing the quality of the solution generators.

The evaluation strategy goes as follows. Let $b = (\mathcal{D}, Q_b, A_b)$ be a *baseline benchmark*, where \mathcal{D} is an RDF dataset, Q_b is a set of keyword queries, and A_b defines the correct answers for the queries in Q_b . The strategy is to construct a synthetic

benchmark $s = (\mathcal{D}, Q_s, A_s)$ for the same dataset \mathcal{D} , using the proposed method, where $Q_b \cap Q_s \neq \emptyset$ and A_s contains, for each keyword query in Q_s , a list of solution generators over \mathcal{D} , synthesized using an implementation of Algorithm 1. Then, for each keyword query in $Q_b \cap Q_s$, we compare the answers in A_b with the solution generators in A_s , as explained in what follows, it is the critical point of the evaluation. Note that we have to guarantee that $Q_b \cap Q_s \neq \emptyset$ as otherwise, the benchmark comparison would have a vacuous effect. Rather than using the keyword query generation method discussion in Section 5, we manually selected keyword queries from the baseline benchmarks to include in the synthetic benchmarks, as also discussed in what follows.

As baselines, we adopted a benchmark for RDF-KwS based on Coffman’s [6] and Dosso’s [7] benchmarks. Coffman’s benchmark was created to evaluate keyword search systems over relational databases and is based on data and schemas of relational samples of IMDb, Mondial, and Wikipedia. Each database has 50 keyword queries and their correct answers. Dosso and Silvello [7] used three real RDF datasets, LinkedMDB, IMDb, and DBpedia, and two synthetic RDF datasets, The Lehigh University Benchmark – LUBM, and The Berlin SPARQL Benchmark – BSBM.

As for the datasets, we chose triplications of relational versions of the full Mondial and IMDb datasets, and not just samples as in Coffman’s benchmark, and the RDF datasets BSBM, LUBM, and DBpedia from Dosso’s benchmark. For each such RDF dataset, we computed the infoRank scores [19].

We selected the keyword queries of the synthetic benchmarks as follows. Quite a few keyword queries in Coffman’s benchmark are *simple queries* in that their expected answers are single entities. Since these queries do not explore the complexity of the graph structure of the RDF datasets, we disregarded them for IMDb and Mondial. We used the keyword queries for BSBM, LUBM, and DBpedia as defined in Dosso’s benchmark. In total, we used 35 keyword queries for IMDb and 24 for Mondial from Coffman’s benchmark, and 50 keyword queries for DBpedia, 14 for LUBM, and 13 for BSBM, from Dosso’s benchmark.

Finally, we ran an implementation of Algorithm 1 to obtain a list of solution generators, for each of the selected keyword queries. The parameters described in Section 6 were set as $\sigma_1 = 0, \sigma_2 = 5, \sigma_3 = 1.0, \sigma_4 = 0.5, \sigma_5 = 0.2, \sigma_6 = 0.2$, for all datasets.

The above process resulted in 5 synthetic benchmarks using IMDb, Mondial, BSBM, LUBM, and DBpedia. The RDF datasets are available at <https://doi.org/10.6084/m9.figshare.11347676.v3>, and the implementation of the Algorithm 1, the keyword queries, the solution generators, and statistics are available at <https://doi.org/10.6084/m9.figshare.16598813.v1>.

We now compared the baseline benchmark with the corresponding synthetic benchmark for each of the five RDF datasets. Consider the following questions (where \mathcal{K} is a keyword query of both the baseline benchmark and the corresponding synthetic benchmark, as explained above):

- Q1. What is the total number $s_{\mathcal{K}}$ of answers expressed by the solution generators for \mathcal{K} in the synthetic benchmark?
- Q2. What is the total number $bs_{\mathcal{K}}$ of answers of \mathcal{K} , defined in the baseline benchmark, that are expressed by the solution generators for \mathcal{K} in the synthetic benchmark?

Q3. What is the total number $sn_{\mathcal{K}}$ of answers expressed by the solution generators for \mathcal{K} in the synthetic benchmark that are not defined in the baseline benchmark?

Let $B_{\mathcal{K}}$ be the set of answers for \mathcal{K} defined in the baseline benchmark and $b_{\mathcal{K}} = |B_{\mathcal{K}}|$.

To address these questions, one has to compute the set $S_{\mathcal{K}}$ of answers for \mathcal{K} that the solution generators for \mathcal{K} express. It depends on the exact notion of answer one is adopting. For example, the set of minimal answers can be estimated by counting the minimal Steiner Trees [22, 8] of a solution generator \mathcal{SG} whose terminal nodes are the set of seeds of \mathcal{SG} . To compute $|B_{\mathcal{K}} \cap S_{\mathcal{K}}|$, one has to test, for each answer $\mathcal{A} \in B_{\mathcal{K}}$, if there is some solution generator for \mathcal{K} that expresses \mathcal{A} . Hence, we have that

- $s_{\mathcal{K}} = |S_{\mathcal{K}}|$
- $bs_{\mathcal{K}} = |B_{\mathcal{K}} \cap S_{\mathcal{K}}|$
- $sn_{\mathcal{K}} = |S_{\mathcal{K}} - B_{\mathcal{K}}| = |S_{\mathcal{K}}| - |B_{\mathcal{K}} \cap S_{\mathcal{K}}|$

Column **#Ms** of Table 5 shows the total number of minimal answers expressed by the solution generators for \mathcal{K} that are not defined in the original benchmarks.

Q1 and Q2 lead to an interesting discussion. Consider that the baseline benchmarks are keyword search systems to be evaluated against the synthetic benchmarks. Then, one can compute the precision of the baseline benchmark for \mathcal{K} against the equivalent synthetic benchmark as $p_{\mathcal{K}} = bs_{\mathcal{K}}/b_{\mathcal{K}}$. The larger $p_{\mathcal{K}}$ is, the larger will be the number of correct answers for \mathcal{K} , in the baseline benchmark, that the solution generators express. Likewise, one can compute the recall of the baseline benchmark for \mathcal{K} against the equivalent synthetic benchmark as $r_{\mathcal{K}} = bs_{\mathcal{K}}/s_{\mathcal{K}}$.

Table 5: Benchmarks statistics obtained for Mondial, IMDb, and DBpedia.

Datasets	Sample Keyword Queries	#Seeds	Precision	#Sol. Generators	#Ms
Mondial	niger country	4	1.00	4	23
	haiti religion	2	1.00	1	2
	mongolia china	4	1.00	2	3
	lebanon syria	5	1.00	3	10
	poland cape verde organization	5	0.82	4	132
	rhein germany province	5	0.50	2	82
	OVERALL AVERAGE			0.91	5.00
IMDb	Johnny Depp Actor	5	1.00	12	46
	Will Smith Male	5	1.00	6	21
	Atticus Finch Movie	5	1.00	10	35
	russell crowe gladiator character	5	0.50	11	21
	sean connery ian fleming work	5	0.11	10	27
	OVERALL AVERAGE			0.52	5.51
DBpedia	Captain America creator notable works	5	1.00	5	47
	Canada Capital	5	1.00	3	57
	governor of Texas	5	1.00	5	58
	Francis Ford Coppola film director	5	1.00	11	61
	mayor of new york city	5	0.00	2	9
	NASA launchpad	5	0.00	3	5
	OVERALL AVERAGE			0.58	8.22

Table 5 summarizes statistics for Mondial, IMDb, and DBpedia. For sample keyword queries (to save space), it shows the number of retrieved seeds, the precision

values that the baseline benchmarks achieved, the number of solution generators obtained from the seeds, and the number of minimal answers expressed by the solution generators that are not defined in the baseline benchmarks. For example, for the keyword query $\mathcal{K} = \{\text{"niger"}, \text{"country"}\}$ from Mondial, the algorithm selected four seeds: the country Niger, the province Niger, the river Niger, and the class Country. Then, it computed four solution generators: 1) with all seeds; 2) with all seeds, except the class Country; 3) with two seeds, the class Country and the node Niger, which is an instance of class Country; and 4) with only the class Country.

The Overall Averages can be interpreted as the percentage of the correct answers, defined in the baseline benchmarks, that the solution generators express - 91% for Mondial, 52% for IMDB, and 58% for DBpedia - which is quite reasonable for synthetic benchmarks.

Finally, we remark that, for the synthetic datasets (BSBM and LUBM), Algorithm 1 precisely found the correct answers listed in Dosso's benchmark.

8 Conclusions

One of the main issues in developing RDF keyword search algorithms is the lack of appropriate benchmarks. This article then proposed an offline method to construct benchmarks for RDF keyword search algorithms. The method automatically computes sets of keyword queries and their correct answers over a given RDF dataset. It circumvents the combinatorial nature of generating correct answers by pruning the search space, following four heuristics based on the concepts of seeds and solution generators. The proposed heuristics introduce flexibilization points of the method that enable the construction of different benchmarks according to the purpose.

The article proceeded to describe synthetic benchmarks for IMDB, Mondial, BSBM, LUBM, and DBpedia. The experiments compared the synthetic benchmarks with baseline benchmarks. They showed that the obtained solution generators express most of the correct answers defined in the baseline benchmarks but express many more answers for IMDB, Mondial, and DBpedia and precisely the defined answers for the synthetic datasets, BSBM and LUBM.

As future work, we plan to fine-tune Algorithm 1 to improve the results summarized in Table 5. We also plan to modify the proposed method to construct training datasets with Natural Language queries over complex RDF datasets.

Acknowledgements This work was partly funded by FAPERJ under grant E-26/202.818/2017; by CAPES under grants 88881.310592-2018/01, 88881.134081/2016-01, and 88882.164913/2010-01 and by CNPq under grant 302303/2017-0. We are grateful to João Guilherme Alves Martinez for helping with the experiments.

Conflict of interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

Appendix A Queries for $\mathcal{K}_{\text{off-Q43}} = \{\text{"mauritius"}, \text{"india"}\}$

A.1 Answer graph pattern

```

1 {
2   BIND (:IND as ?s1) # sets India's URI
3   ?s1 a :Country.
4   ?s2 a :Country.
5   ?s3 a :Organization.
6   ?s1 :isMember ?s3.
7   ?s2 :isMember ?s3.
8   FILTER (?s1!=?s2 && ?s1!=?s3 && ?s2!=?s3)
9 }

```

A.2 Query for retrieving property values

```

1 SELECT ?l1 ?l2 ?l3 ?l4 ?l5 ?l6 ?v1 ?v2 ?v3
2 WHERE {
3   {
4     BIND (:IND AS ?s1) # sets India's URI
5     ?s1 a :Country.
6     ?s2 a :Country.
7     ?s3 a :Organization.
8     ?s1 :isMember ?s3.
9     ?s2 :isMember ?s3.
10    FILTER (?s1!=?s2 && ?s1!=?s3 && ?s2!=?s3)
11  }
12  :Country rdfs:label ?l1.
13  :Organization rdfs:label ?l2.
14  :isMember rdfs:label ?l3.
15
16  OPTIONAL
17  {SELECT ?s1 (group_concat(DISTINCT ?l) AS ?l4)
18  WHERE {?s1 ?p ?o. ?p rdfs:label ?l.}
19  GROUP BY ?s1}
20  OPTIONAL
21  {SELECT ?s2 (group_concat(DISTINCT ?l) AS ?l5)
22  WHERE {?s2 ?p ?o. ?p rdfs:label ?l.}
23  GROUP BY ?s2}
24  OPTIONAL
25  {SELECT ?s3 (group_concat(DISTINCT ?l) AS ?l6)
26  WHERE {?s3 ?p ?o. ?p rdfs:label ?l.}
27  GROUP BY ?s3}
28
29  {SELECT ?s1 (group_concat(?o) AS ?v1)
30  WHERE {?s1 ?p ?o FILTER isLiteral(?o)}
31  GROUP BY ?s1}
32  OPTIONAL
33  {SELECT ?s2 (group_concat(?o) AS ?v2)
34  WHERE {?s2 ?p ?o FILTER isLiteral(?o)}
35  GROUP BY ?s2}
36  {SELECT ?s3 (group_concat(?o) AS ?v3)
37  WHERE {?s3 ?p ?o FILTER isLiteral(?o)}
38  GROUP BY ?s3}

```

```
39 }
```

A.3 Query to generate the Coffman’s ground truth

```

1  INSERT {
2      GRAPH ?g {
3          ?s1 :isMember ?s3; ?p1 ?v1.
4          ?s2 :isMember ?s3; ?p2 ?v2.
5      }
6  }
7  WHERE {
8      BIND (UUID() AS ?g)
9      {
10         ?s1 a :Country.
11         ?s2 a :Country.
12         ?s3 a :Organization.
13         ?s1 :isMember ?s3.
14         ?s2 :isMember ?s3.
15         FILTER (?s1!=?s2 && ?s1!=?s3 && ?s2!=?s3)
16     }
17     ?s1 ?p1 ?v1.
18     ?s2 ?p2 ?v2.
19
20     FILTER (REGEX(?v1,"India","i") &&
21             REGEX(?v2,"Mauritius","i"))
22 }
```

Appendix B Queries for $\mathcal{K}_{\text{Coff-Q30}} = \{\text{“polland”, “spoken”, “language”}\}$

B.1 Answer graph pattern

```

1  {
2      BIND (:PL as ?s1) # sets Poland’s URI
3      ?s1 a :Country.
4      ?s2 a :SpokenBy.
5      ?s2 :languageInfo- ?s1.
6      FILTER (?s1!=?s2)
7  }
```

B.2 Query for retrieving property values

```

1  SELECT ?l1 ?l2 ?l3 ?l4 ?l5 ?v1 ?v2
2  WHERE {
3      {
4          BIND (:PL AS ?s1) #sets Poland’s URI
5          ?s1 a :Country.
6          ?s2 a :SpokenBy.
7          ?s2 :languageInfo- ?s1.
```

```

8      FILTER (?s1!=?s2)
9    }
10   :Country rdfs:label ?l1.
11   :SpokenBy rdfs:label ?l2.
12   :languageInfo- rdfs:label ?l3.
13
14   OPTIONAL
15   {SELECT ?s1 (group_concat(DISTINCT ?l) AS ?l4)
16   WHERE {?s1 ?p ?o. ?p rdfs:label ?l.}
17   GROUP BY ?s1}
18   OPTIONAL
19   {SELECT ?s2 (group_concat(DISTINCT ?l) AS ?l5)
20   WHERE {?s2 ?p ?o. ?p rdfs:label ?l.}
21   GROUP BY ?s2}
22
23   {SELECT ?s1 (group_concat(?o) AS ?v1)
24   WHERE {?s1 ?p ?o FILTER isLiteral(?o)}
25   GROUP BY ?s1}
26   {SELECT ?s2 (group_concat(?o) AS ?v2)
27   WHERE {?s2 ?p ?o FILTER isLiteral(?o)}
28   GROUP BY ?s2}
29 }

```

B.3 Query to generate the Coffman's ground truth

```

1  INSERT {
2    GRAPH ?g {
3      ?s2 a ?cls.
4      ?s2 :languageInfo- ?s1.
5      ?s2 ?p2 ?s3.
6      ?s1 ?p1 ?v1.
7      ?cls rdfs:label ?l2.
8    }
9  }
10 WHERE {
11   BIND (UUID() AS ?g)
12   {
13     ?s1 a :Country.
14     ?s2 a ?cls.
15     ?s2 :languageInfo- ?s1.
16     FILTER (?s1!=?s2)
17   }
18   ?s1 ?p1 ?v1.
19   ?cls rdfs:label ?l2.
20   ?s2 ?p2 ?s3.
21   ?p2 rdfs:label ?l5.
22
23
24   FILTER (REGEX(?v1,"Poland","i") &&
25           REGEX(?l2,"Spoken","i") &&
26           REGEX(?l5,"language","i"))
27 }

```

Appendix C Queries for $\mathcal{K}_{\text{Coff-Q27}} = \{\text{"nigeria"}, \text{"gdp"}\}$

C.1 Answer graph pattern

```

1 {
2     BIND (:WAN AS ?s1) #sets Nigeria's URI
3     ?s1 a :Country.
4 }

```

C.2 Query for retrieving property values

```

1 SELECT ?l1 ?l2 ?v1
2 WHERE {
3     {
4         BIND (:WAN AS ?s1) # sets Nigeria's URI
5         ?s1 a :Country.
6     }
7     :Country rdfs:label ?l1.
8
9     {SELECT ?s1 (group_concat(DISTINCT ?l) AS ?l2)
10    WHERE {?s1 ?p ?o. ?p rdfs:label ?l.}
11    GROUP BY ?s1}
12
13    {SELECT ?s1 (group_concat(?o) AS ?v1)
14    WHERE {?s1 ?p ?o FILTER isLiteral(?o)}
15    GROUP BY ?s1}
16 }

```

C.3 Query to generate the Coffman's ground truth

```

1 INSERT {
2     GRAPH ?g {
3         ?s1 ?p1 ?v1.
4         ?s1 ?p2 ?v2.
5     }
6 }
7 WHERE {
8     BIND (UUID() AS ?g)
9     {
10        ?s1 a :Country.
11    }
12    ?s1 ?p1 ?v1.
13    ?s1 ?p2 ?v2.
14    ?p2 rdfs:label ?l2.
15
16    FILTER (REGEX(?v1,"Nigeria","i") &&
17            REGEX(?l2,"gdp","i"))
18 }

```

References

1. Balog K, Neumayer R (2013) A test collection for entity search in DBpedia. In: Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, New York, NY, USA, pp 737–740, DOI 10.1145/2484028.2484165
2. Bast H, Buchhold B, Haussmann E (2016) Semantic Search on Text and Knowledge Bases. *Foundations and Trends in Information Retrieval* 10(1):119–271, DOI 10.1561/15000000032
3. Batista Neves A, André Paes Leme LP, Torres Izquierdo Y, Antonio Casanova M (2021) Automatic Construction of Benchmarks for RDF Keyword Search Systems Evaluation. In: Proceedings of the 23rd International Conference on Enterprise Information Systems (ICEIS'21), DOI 10.5220/0010519401260137
4. Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S (2002) Keyword searching and browsing in databases using BANKS. In: Proceedings of the 18th International Conference on Data Engineering (ICDE'02), IEEE Comput. Soc, pp 431–440, DOI 10.1109/ICDE.2002.994756
5. Bizer C, Schultz A (2009) The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems* 5(2):1–24, DOI 10.4018/jswis.2009040101
6. Coffman J, Weaver AC (2010) A framework for evaluating database keyword search strategies. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10), ACM Press, New York, New York, USA, p 729, DOI 10.1145/1871437.1871531
7. Dosso D, Silvello G (2020) Search Text to Retrieve Graphs: A Scalable RDF Keyword-Based Search System. *IEEE Access* 8:14089–14111, DOI 10.1109/ACCESS.2020.2966823
8. Dourado MC, de Oliveira RA, Protti F (2009) Generating all the Steiner trees and computing Steiner intervals for a fixed number of terminals. *Electronic Notes in Discrete Mathematics* 35:323–328, DOI 10.1016/j.endm.2009.11.053
9. Dubey M, Banerjee D, Abdelkawi A, Lehmann J (2019) LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia. In: Proceedings of the 18th International Semantic Web Conference (ISWC'19), pp 69–78, DOI 10.1007/978-3-030-30796-7{_}5
10. Elbassuoni S, Blanco R (2011) Keyword search over RDF graphs. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11), ACM Press, New York, New York, USA, p 237, DOI 10.1145/2063576.2063615
11. García GM, Izquierdo YT, Menendez ES, Dartayre F, Casanova MA (2017) RDF Keyword-based Query Technology Meets a Real-World Dataset. In: Proceedings of the 20th International Conference on Database Theory (ICDT'17), pp 656–667, DOI 10.5441/002/edbt.2017.86
12. Guo Y, Pan Z, Heflin J (2005) LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics* 3(2-3):158–182, DOI 10.1016/j.websem.2005.06.005

13. Han S, Zou L, Yu JX, Zhao D (2017) Keyword Search on RDF Graphs - A Query Graph Assembly Approach. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge (CIKM'17), ACM, New York, NY, USA, pp 227–236, DOI 10.1145/3132847.3132957
14. Hristidis V, Papakonstantinou Y (2002) Discover: Keyword Search in Relational Databases. In: Proceedings of the 28th International Conference on Very Large Databases (VLDB'02), Elsevier, pp 670–681, DOI 10.1016/B978-155860869-6/50065-2
15. Izquierdo YT, García GM, Menendez ES, Casanova MA, Dartayre F, Levy CH (2018) QUIOW: A Keyword-Based Query Processing Tool for RDF Datasets and Relational Databases. In: Proceedings of the 30th International Conference on Database and Expert Systems Applications (DEXA'18), vol 11030 LNCS, pp 259–269, DOI 10.1007/978-3-319-98812-2_{_}22
16. Kimelfeld B, Sagiv Y (2008) Efficiently enumerating results of keyword search over data graphs. *Information Systems* 33(4-5):335–359, DOI 10.1016/j.is.2008.01.002
17. Le W, Li F, Kementsietsidis A, Duan S (2014) Scalable Keyword Search on Large RDF Data. *IEEE Transactions on Knowledge and Data Engineering* 26(11):2774–2788, DOI 10.1109/TKDE.2014.2302294
18. Lin XQ, Ma ZM, Yan L (2018) RDF keyword search using a type-based summary. *Journal of Information Science and Engineering* 34(2):489–504, DOI 10.6688/JISE.201803_{_}34(2).0011
19. Menendez ES, Casanova MA, Paes Leme LAP, Boughanem M (2019) Novel Node Importance Measures to Improve Keyword Search over RDF Graphs. In: Proceedings of the 31st International Conference on Database and Expert Systems Applications (DEXA'19), vol 11707, pp 143–158, DOI 10.1007/978-3-030-27618-8_{_}11
20. Minack E, Siberski W, Nejd W (2009) Benchmarking Fulltext Search Performance of RDF Stores. In: Proceedings of the 6th European Semantic Web Conference (ESWC'09), vol 5554 LNCS, pp 81–95, DOI 10.1007/978-3-642-02121-3_{_}10
21. Nunes BP, Herrera J, Taibi D, Lopes GR, Casanova MA, Dietze S (2014) 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), pp 461–466, DOI 10.1007/978-3-319-11955-7_{_}67
22. Oliveira PSd, Da Silva A, Moura E, De Freitas R (2020) Efficient Match-Based Candidate Network Generation for Keyword Queries over Relational Databases. *IEEE Transactions on Knowledge and Data Engineering* pp 1–1, DOI 10.1109/TKDE.2020.2998046
23. Oliveira Filho AdC (2018) Benchmark para métodos de consultas por palavras-chave a bancos de dados relacionais. Tech. rep., Universidade Federal de Goiás, Goiás
24. Pound J, Mika P, Zaragoza H (2010) Ad-hoc Object Retrieval in the Web of Data. In: Proceedings of the 19th International Conference on World Wide Web, pp 771–780

25. Rihany M, Kedad Z, Lopes S (2018) Keyword search over RDF graphs using wordnet. In: Proceedings of the 1st International Conference on Big Data and Cyber-Security Intelligence (BDCSIntell'18), vol 2343, pp 75–82
26. Tran T, Wang H, Rudolph S, Cimiano P (2009) Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In: Proceedings of the 25th International Conference on Data Engineering (ICDE'09), IEEE, pp 405–416, DOI 10.1109/ICDE.2009.119
27. Trivedi P, Maheshwari G, Dubey M, Lehmann J (2017) LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs. In: Proceedings of the 16th International Semantic Web Conference (ISWC'17), pp 210–218, DOI 10.1007/978-3-319-68204-4_{_}22
28. Wen Y, Jin Y, Yuan X (2018) KAT: Keywords-to-SPARQL Translation Over RDF Graphs. In: Proceedings of the 23rd International Conference on Database Systems for Advanced Applications (DASFAA'18), vol 10827 LNCS, pp 802–810, DOI 10.1007/978-3-319-91452-7_{_}51
29. Zenz G, Zhou X, Minack E, Siberski W, Nejd W (2009) From keywords to semantic queries—Incremental query construction on the semantic web. *Journal of Web Semantics* 7(3):166–176, DOI 10.1016/j.websem.2009.07.005
30. Zheng W, Zou L, Peng W, Yan X, Song S, Zhao D (2016) Semantic SPARQL similarity search over RDF knowledge graphs. In: Proceedings of the 42nd VLDB (VLDB'16), vol 9, pp 840–851, DOI 10.14778/2983200.2983201
31. Zhou Q, Wang C, Xiong M, Wang H, Yu Y (2007) SPARK: Adapting Keyword Query to Semantic Search. In: Proceedings of the 6th International Semantic Web Conference (ISWC'07), Busan, Korea, vol 4825 LNCS, pp 694–707, DOI 10.1007/978-3-540-76298-0_{_}50