

***QUIOW*: A Keyword-based Query Processing Tool for RDF Datasets and Relational Databases**

Yenier T. Izquierdo^{1,2}, Grettel M. García^{1,2}, Elisa S. Menendez¹,
Marco A. Casanova^{1,2}, Frederic Dartayre², Carlos H. Levy²

¹Department of Informatics – Pontifical Catholic University of Rio de Janeiro, RJ, Brazil

²Instituto TecGraf – Pontifical Catholic University of Rio de Janeiro, RJ, Brazil

{yizquierdo,ggarcia,emenendez,casanova}@inf.puc-rio.br,

{fdartayre,levy}@tecgraf.puc-rio.br

Abstract. This paper describes a keyword-based query processing tool, called *QUIOW*, deployed in two distinct environments: RDF datasets with schemas and relational databases. The tool first translates a keyword-based query into an abstract query, and then compiles the abstract query into a concrete (SPARQL or SQL) query such that each result of concrete query is an answer for the keyword-based query. The tool explores the schema to avoid user intervention during the translation process. The paper includes extensive experiments to compare keyword-based query processing in the two environments, using a full version of IMDb – The Internet Movies Database and the Mondial database.

Keywords: Keyword search; SQL; SPARQL; Relational model; RDF.

1 Introduction

Database applications that offer keyword search free the users from filling “boxes” with exact data by compiling keyword-based queries to the query language supported, and by ranking the results. In fact, keyword search applications over relational databases have been studied for quite some time. More recently, examples of such applications designed for RDF datasets have emerged.

In this paper, we describe *QUIOW*, a keyword-based query processing tool deployed in two distinct environments: RDF datasets with schemas and relational databases. The tool first translates a keyword-based query into an abstract query, and then compiles the abstract query into a concrete (SPARQL or SQL) query such that each result of concrete query is an answer for the keyword-based query. The tool explores the schema to avoid user intervention during the translation process, and to minimize the number of joins in a query. The implementation of the tool is engineered to work with different RDF stores and relational DBMSs. The current implementation supports Oracle 12c, for both the RDF and relational environments.

The paper also covers expensive experiments that use RDF and relational versions of the Mondial database – compiled from geographical Web data sources and a complete variation of IMDb – The Internet Movies Database, which includes descriptions

of artists, movies, documentaries, TV series, and even computer games. The experiments with Mondial and IMDb extend the Coffman and Weaver’s benchmark [5] and were conducted to fully assess the performance of the tool in the two environments.

The *QUIOW* tool evolved from an earlier implementation [7], which proved efficient, but only worked with RDF datasets and suffered from the problem of maintaining the RDF dataset synchronized with the source relational database. The *QUIOW* tool differs from the earlier tool in four important aspects. First, it includes a backtracking mechanism to generate alternative translations for a keyword-based query, which expands the set of answers returned for the keyword-based query, since the answers now reflect the result of executing several SPARQL (or SQL) queries. Second, the tool supports both the RDF and the relational environments. Third, the negation operator is included in the property filter options. Lastly, the tool is engineered to work with different RDF stores and relational DBMSs.

The contributions of this paper can be summarized as follows. First, the paper describes a keyword-based query processing tool, deployed in two distinct environments: RDF datasets with schemas and relational databases. This is the first tool with these characteristics to be described in the literature, to the best of our knowledge. Second, it includes extensive experiments using Mondial and IMDb to help assess the performance of the tool and compare keyword-based query processing in two environments.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 introduces the keyword translation algorithm the *QUIOW* tool uses. Section 4 describes extensive experiments to compare keyword-based query processing over RDF datasets and relational databases and to assess the performance of the tool. Finally, Section 5 presents the conclusions and future work.

2 Related Work

Recent surveys of keyword-based query processing tools over relational databases and RDF datasets can be found in [3,16]. Early relational keyword-based query processing tools [1,2,10,11] explored the foreign/primary keys declared in the relational schema to compile a keyword-based query into an SQL query with a minimal set of join clauses, based on the notion of candidate networks (CNs). This approach was also adopted in recent tools [4,13]. In particular, *QUEST* [4] explores the structure of the conceptual schema to synthesize an SQL query, based on a Steiner tree that captures a minimum set of joins. Tastier [13] included the user in the keyword-based query processing loop and provided context-sensitive auto-completion of keyword queries, via specialized data structures that index the database. Coffman and Weaver [5] presented a qualitative evaluation of several relational keyword-based query processing tools, using a benchmark, which consists of a simplified version of IMDb, a subset of Wikipedia, and a subset of the Mondial dataset, and a set of 50 keyword queries for each database.

An RDF keyword-based query processing tool can be categorized as *schema-based*, when it exploits the RDF schema to compile a keyword-based query into a SPARQL query, or as *graph-based*, when it directly operates on the RDF dataset. We provide a brief review of each of these approaches.

We start with RDF schema-based approaches. Han et al. [9] described an algorithm that assembles a query from the keywords and experiments with DBpedia and Freebase.

QUICK [17] translates keyword-based queries to SPARQL queries with the help of the users, who choose a set of intermediate queries, which the tool ranks and executes. Gkirtzou et al. [8] described a method to generate candidate SPARQL queries, with natural language descriptions, to help users decide which query to execute.

As for graph-based tools, SPARK [19] 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. The most likely SPARQL query is selected using a probabilistic ranking model that incorporates the quality of the mapping and the structure of the query is proposed. Tran et al. [15] combined the idea of generating summary graphs for the RDF graph, using the class hierarchy, to generate and rank candidate SPARQL queries. Le et al. [12] also proposed to process keyword queries using a summarization algorithm. Zheng et al. [18] also adopted a pattern-based approach. Elbassuoni and Blanco [6] described a technique to retrieve a set of sub-graphs that match the keywords, and to rank them based on statistical language models.

3 The Keyword Search Tool

3.1 The Keyword Translation Algorithm

The keyword translation algorithm: (1) accepts a keyword-based query K over an RDF dataset T (or a relational database T), together with its RDF (or relational) schema S ; (2) finds matches with the keywords in K ; (3) creates an abstract query by exploring the keyword matches found and the schema S ; (4) compiles the abstract query into a SPARQL (or SQL query), which is then executed.

Schema graph. The algorithm uniformly treats the RDF or relational schema S as a labelled schema multigraph $G_S=(N_S,E_S,EL_S)$. In an RDF environment, N_S are the classes, and EL_S labels arcs with object properties or with `rdfs:subsetOf` in E_S , as in the RDF graph induced by S . In a relational environment, N_S are the relation scheme names, and EL_S labels arcs in E_S with foreign key names, as in the multigraph induced by S .

Computing keyword matches. Consider first the RDF environment. The algorithm starts by computing: (1) a set of classes and properties in S whose metadata (i.e., labels and descriptions) match keywords in K ; (2) a set of property/value pairs (p,v) such that there is a triple (s,p,v) in T such that v matches a keyword in K .

The algorithm organizes the result of the matches as a set of nucleuses. Each *nucleus* is a triple (c,PR,VA) , where c is a class, PR is a possibly empty list of properties, and VA is a possibly empty list of property/value pairs, such that: (1) if PR and VA are empty, c must be the result of a metadata match; (2) each property in PR has c as domain and is the result of a metadata match; (3) each property/value pair (p,v) in VA is such that the domain of p is c and (p,v) is the result of a data match.

For the relational environment, the algorithm proceeds almost exactly as for the RDF case. It computes: (1) a set of relation scheme and attribute in S whose metadata (i.e., names and descriptions) match keywords in K ; (2) a set of attribute/value pairs whose values match keywords in K . A nucleus is again a triple (c,PR,VA) , where c is a relation scheme name, PR is a possibly empty list of attributes, and VA is a possibly empty list of attribute/value pairs, such that: (1) if PR and VA are empty, c is the result of a

metadata match; (2) each element in PR is an attribute of c and is the result of a metadata match; (3) each attribute/value pair (p,v) in VA is such that p is an attribute of c and (p,v) reflects a data match.

In either case, RDF or relational, the matching process results in a set of nucleuses.

Synthesizing an abstract query. The next stage is to synthesize an abstract query that captures the keyword-based query. It depends on the schema multigraph and the set of nucleuses that the matching process outputs, independently of the environment. To synthesize an abstract query, the translation algorithm implements two heuristics, called the *scoring* and the *minimization* heuristics.

Briefly, the scoring heuristic: (1) considers how good a match is, say the keyword “south” matches the literal “south” better than the literal “South Africa”; (2) assigns a higher score to metadata matches, on the grounds that, if the user specifies a keyword, say “Desert”, that matches both a class label (or relation scheme name), say “Desert”, and a property (or attribute) value of an instance (or a tuple), say the location “Sahara Desert”, then the user is probably more interested in the class (scheme) labelled “Desert” than the specific instance “Sahara Desert”; (3) assigns a higher score to nucleuses that cover a larger number of keywords. The heuristic is formalized by defining a *score function* for the nucleuses [7].

The minimization heuristic tries to generate minimal answers, in two stages. Ideally, it should try to find the smallest set of nucleuses that covers the largest set of keywords and that has the largest combined score. However, this is an NP-complete problem. The first stage of the minimization heuristic then implements a greedy algorithm that prioritizes the nucleuses with the largest scores that cover a large subset of K . The second stage of the minimization heuristic then connects the classes (or relation scheme) in such nucleuses, using a small number of equijoins. This is equivalent to generating a Steiner tree ST of the schema graph that covers the classes (or relation scheme) of the prioritized nucleuses. Finally, the algorithm uses the edges of ST to generate join clauses, and the nucleuses to generate selection clauses of the abstract query.

Compiling the abstract query into a concrete query. The final stage of the translation algorithm is to compile the abstract query into a concrete SPARQL or SQL query for the underlying RDF store or relational DBMS. Section 4.2 illustrates this process.

Execution. The tool then executes the concrete query to generate representations of answers to the keyword-based query, which are then passed to the user.

3.2 An Example

This section discusses how the keyword translation algorithm works, using Mondial, whose ER-Diagram is available at <https://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-ER.pdf>. Figure 1(a) shows the keyword-based query $K = \{Country, Province, Desert, Atacama\}$. Figure 1(b) depicts the structure of an abstract query that corresponds to a Steiner tree of the schema graph of Mondial, where:

- The nodes correspond to classes *Country*, *Province*, *Desert*, and *Geo_Desert*; the first three nodes correspond to nucleuses and the fourth node just completes the tree to link the classes *Province* and *Desert*.
- The arcs represent object properties between such classes.

The classes that correspond to nucleuses reflect the following matches:

- *Country*, *Province*, and *Desert* have an exact metadata match with the labels of classes *Country*, *Province* and *Desert*.
- *Atacama* matches a value of property *Label* whose domain is the class *Desert*.

Note that the keyword *Desert* is ambiguous since it matches with the labels of classes *Desert* and *Geo_Desert*, but the translation algorithm selects the class *Desert* because the corresponding nucleus has the highest score. Also, the match found with the instance of class *Desert* whose label is *Atacama* does not need to be shown in the abstract query because that information is in the nucleus data.

$K = \{Country, Province, Desert, Atacama\}$

(a) Keyword-based query

```

1. SELECT DISTINCT ?C0 ?C1 ?C2
2. WHERE{
3. ?I_C0 rdf:type <http://www.swtech.org/mondial/Country> .
4. ?I_C1 rdf:type <http://www.swtech.org/mondial/Desert> .
5. ?I_C2 rdf:type <http://www.swtech.org/mondial/Geo_Desert> .
6. ?I_C3 rdf:type <http://www.swtech.org/mondial/Province> .
7. ?I_C2 <http://www.swtech.org/mondial/Geo_Desert#Province> ?I_C3 .
8. ?I_C3 <http://www.swtech.org/mondial/Province#Code> ?I_C0 .
9. ?I_C2 <http://www.swtech.org/mondial/Geo_Desert#Desert> ?I_C1
10. FILTER (orrdf:textContains(?C1, "{atacama}", 0) )
11. ?I_C0 rdfs:label ?C0 .
12. ?I_C1 rdfs:label ?C1 .
13. ?I_C3 rdfs:label ?C2 }

```

(c) SPARQL Query



(b) Abstract Query

```

1. SELECT Desert.META_REPCOL,
2. Desert.NAME,
3. Country.META_REPCOL,
4. Country.CODE,
5. Province.META_REPCOL,
6. Province.NAME,
7. Province.COUNTRY
8. FROM Desert, Country, Province, Geo_Desert
9. WHERE
10. ((contains(Desert.META_REPCOL,'{atacama}', 0) > 0)
11. AND (Geo_Desert.DESERT = Desert.NAME)
12. AND (Geo_Desert.PROVINCE = Province.NAME)
13. AND (Province.COUNTRY = Country.CODE)

```

(d) SQL Query

Fig. 1. Sample SPARQL and SQL queries.

From this abstract query, the translation algorithm generates the SPARQL query shown in Figure 1(c), where:

- Line 1 constructs the result of the query, which consists of the labels of instances of the classes *Country*, *Desert*, and *Province*, respectively, as shown at Figure 2(a).
- Lines 3 to 6 introduce variables that range over instances of the classes that correspond to the nodes of the Steiner tree.
- Lines 7 to 9 represent relationships between the instances, that correspond to the arcs of the Steiner tree.
- Line 10 represents the match with the keyword *Atacama*.
- Lines 11 to 13 translate instances to their labels, which are user-friendly.

The translation algorithm generates the SQL query shown in Figure 1(d), where:

- Lines 1 to 7 capture the primary keys and the META_REPCOL columns of the relation schemes. For each relation, the preparation stage adds a column, META_REPCOL, whose values parallel the instance labels of the RDF version.
- Line 8 corresponds to the nodes of the Steiner tree.
- Line 10 represents the match with the keyword *Atacama*.
- Lines 11 to 13 represent equijoins that correspond to the arcs of the Steiner tree.

Note that in the line 10 of both queries, equivalent contains operators of Oracle Text are used to filter the results by the instance of *Desert* with label *Atacama*. In two cases, the contains operators were configured using the default parameters.

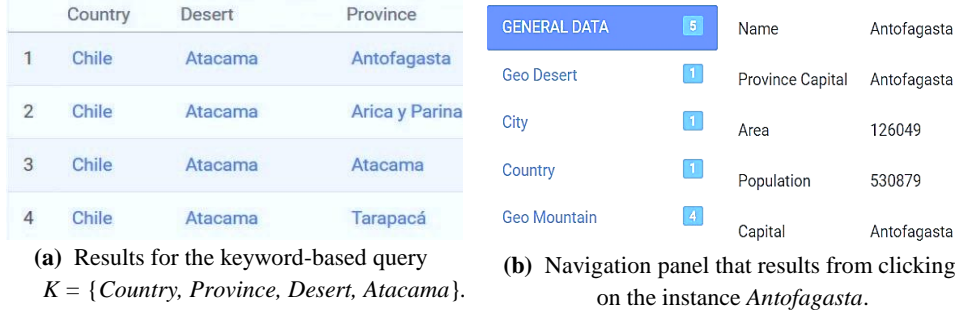


Fig. 2. Sample query results and navigation panel.

Navigation over the results of a keyword-based query. The tool does not return the results of a keyword-based query as a set of RDF triples or as a set of tuples. After several experiments with the users, we decided to present the results of a keyword-based query in a tabular format for both the RDF and the relational environments, combined with facilities to help users explore the results. For example, Figure 2(a) shows the result of the keyword-based query in Figure 1(a). If the user clicks on *Antofagasta*, the tool will display the results shown in Figure 2(b).

In the RDF environment, navigation over the query results is much simpler than in the relational environment, since it directly crawls the RDF graph. Also, navigation leverages on the ability of SPARQL queries to retrieve data and metadata seamlessly.

To support navigation over data about an instance, the tool uses two queries:

- 1) SPARQL query Q_1 :
 1. SELECT ?Class ?Property ?Value ?Label
 2. WHERE {
 3. ?uri ?prop ?Value .
 4. ?prop rdfs:label ?Property .
 5. OPTIONAL { ?Value rdfs:label ?Label } .
 6. ?uri rdfs:type ?c .
 7. ?c rdfs:label ?Class }
- 2) SPARQL query Q_2 :
 1. SELECT ?Class ?Value ?Label
 2. WHERE {
 3. ?Value ?property ?uri .
 4. OPTIONAL { ?Value rdfs:label ?Label } .
 5. ?Value rdfs:type ?c .
 6. ?c rdfs:label ?Class
 7. }

We assume that all classes, instances of classes, and properties have a mandatory label. Consider query Q_1 , and assume that variable $?uri$ in Line 3 binds to instance I . The query retrieves any property and its value that I has (Line 3), the label of such property (Line 4), the label of the value, if it corresponds to an object property value (Line 5), any class that I belongs to (Line 6), and the label of such class (Line 7). Query Q_2 retrieves an instance J that is related to I by an object property (Line 3), the label of J , if it exists (Line 4), a class that J belongs to (Line 5), and the label of such class (Line 6). The results of both queries are then post-processed to generate a navigation panel. Figure 2(b) illustrates the navigation panel that results from clicking on *Antofagasta*.

However, it is not as simple to explore the query results in the relational environment as is in the RDF environment. The tool implements the following strategy:

- (1) Construct a SQL query, using the template:

SELECT pk₁, ..., pk_n, Label, p₁, p₂, ..., p_n FROM T WHERE *<instance filter>*

that queries table T to retrieve the primary keys, instance label, and properties, and whose WHERE clause is a filter defined by the identifier created for the target instance. In our example, the filter is built using the primary key values for instance *Antofagasta*.

- (2) Compute a set JF of tables such that $V \in JF$ iff T has a foreign key to V . For each $V \in JF$, a SQL query with an inner join between T and V and a TARGET clause composed of the primary keys and the label columns of V is compiled and executed.
- (3) Compute a set JT of tables such that $U \in JT$ iff U has a foreign key to T . For each table $U \in JT$, a SQL query with an inner join between T and U and a TARGET clause composed of the primary keys and the label columns of U is compiled and executed.
- (4) Obtain the labels of the properties retrieved in (a), the label of T , and the labels of the tables in $JF \cup JT$.

The data in Figure 2(b) is built by post-processing the results of steps (3) and (4).

In our running example, the generation of the panel in Figure 2(b) required 2 SPARQL queries over the RDF graph, which ran in 0.89s, while it required 9 SQL queries over the relational database, which ran in 2.75s. Thus, navigation over the RDF graph was nearly 3 times faster than in the relational database.

4 Experiments

Experimental Setup. *QUIOW* was implemented as a RESTful Web application developed in Java, and ran on Windows 7 Ultimate, using a quad-core processor Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, 4 GB of RAM. The relational databases and RDF datasets were stored in Oracle 12c, running on a quad-core processor Intel(R) Core(TM) i5 CPU 660 @ 3.33GHz, 7GB of RAM, and 4,096 KB of cache size.

Table 1 shows basic statistics about the RDF datasets and relational databases used in the experiments. We ran the Coffman’s benchmark [5] using the relational databases and their triplified versions of the Mondial dataset (available at <https://www.dbis.informatik.uni-goettingen.de/Mondial/>) and the full and more recent relational version of IMDb (available at <https://sites.google.com/site/ontopiswc13/home/imdb-mo>), which we refer to as Full IMDb to differ it from the Restricted IMDb version used in [5].

We measured the query build time – the time taken to process matches and construct the SQL or SPARQL query, and total elapsed time – the time from the submission of the keyword-based query until the display of the first 75 results.

We used a separated tool to index the relational databases to support keyword search and to triplify the relational databases via a set of relational-to-RDF mappings and to create indexes over the RDF dataset to support keyword search. In this case, we measured the time to index the relational database and to create the RDF dataset.

Table 1. Statistics of Mondial and IMDb Datasets.

RDF Dataset	#Triples		Relational Database	#Objects	
	Mondial	IMDb		Mondial	IMDb
Classes	40	11	# of Relations	40	11
Object properties	62	11	# of Attributes	187	20
Datatype properties	130	25	# of Tuples	40,247	70,520,722
Indexed properties	71	7	Size (in GB)	0.11	60.63
Indexed prop. instances	11,094	12,609,418			
Class instances	43,869	70,520,744			
Object prop. instances	63,652	204,917,673			
Total number of triples	235,387	382,295,213			

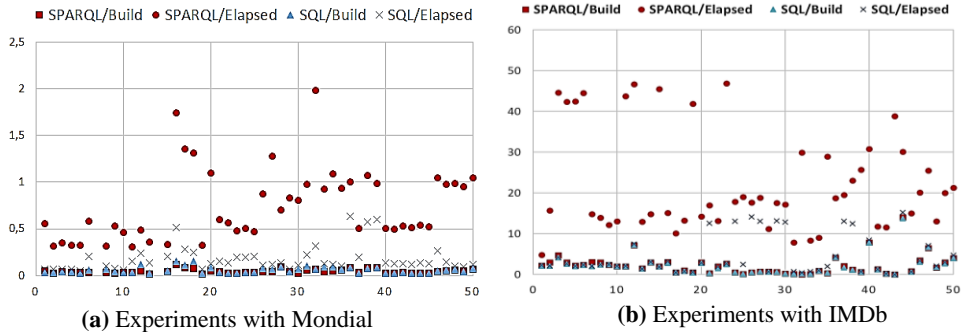


Fig. 3. Build Time and Total Elapsed Time.

Experiments with the Coffman’s Benchmark Datasets. Figure 3 shows the execution times of the keyword-based queries defined in Coffman’s benchmark. The Y-axis represents the query build time and the total elapsed time, in seconds, of each query in the benchmark, numbered 1 to 50 on the X-axis. Note that, for each keyword-based query: the SPARQL total elapsed time (shown as a dot) was always larger than the SQL total elapsed time (shown as a cross), and the SPARQL and the SQL query build times (respectively shown as squares and triangles) were nearly the same (most squares are on top of the triangles).

To reduce ambiguity when using the Full IMDb, and consequently to improve processing time, we surrounded most keywords with quotes. For instance, consider the query $\{denzel\} \{washington\}$. If we treat the keywords separately, we find that $\{denzel\}$ has 670 data matches, while $\{washington\}$ has 23,720. Indeed, “washington” is a very ambiguous keyword, since it matches the name of an actor, city, state, etc. Hence, if we treat the query as “denzel OR washington” we have a total of 23,851 data matches. However, if we treat the query as “denzel AND washington”, we have only 539 data matches. As pointed out in [7], this affects the computation of the scores. Indeed, using quotes, the total elapsed times of the SQL query to compute the value score for class *movie_info* and property *info* was 0.14s. By contrast, without quotes, the total elapsed time was 5.04s. Since we execute queries for all distinct domains and properties that match at least one of the keywords, the total time to create an abstract query in this example was, on average, 30 times faster with quotes. We note that the use of quotes only improved the query build time and did not change the final results.

Table 2 shows the average (Avg), maximum (Max) and minimum (Min) of the total elapsed time and the query build time of the 50 queries in Coffman’s benchmark, for the relational and RDF variations of Mondial and IMDb.

Table 2. Summary of the experiments with Mondial and IMDb datasets.

		Mondial	IMDb	Mondial	IMDb	Mondial	IMDb
		Total Elapsed Time (in sec)		Query Build Time (in sec)		Query Build Time / Total Elapsed Time	
Avg.	Relational	0.147	4.360	0.066	2.128	50.5%	71.2%
	RDF	0.802	22.273	0.047	2.263	6.6%	11.2%
Max.	Relational	0.516	15.279	0.154	13.765	73.6%	98.8%
	RDF	1.982	46.868	0.121	14.016	13.5%	46.4%
Min.	Relational	0.051	0.093	0.021	0.064	19.6%	1.9%
	RDF	0.311	7.895	0.012	0.030	3.3%	0.1%

Preparation and Maintenance. In the relational environment, preparing Mondial to support keyword search took less than a minute, while preparing IMDB took 60 minutes. Thus, re-indexing would be a feasible strategy to maintain the relational version of Mondial, but slow for IMDB. Triplifying Mondial, which is a small database, took 5 minutes; thus, full retriplification would be a feasible strategy to maintain the RDF version. However, triplifying IMDB was very slow, taking 5 hours. This calls for an incremental strategy to maintain complex RDF datasets, such as IMDB.

Quality of the query results. For each keyword search executed, the results obtained were exactly the same in both the RDF and relational environments, as expected, since the construction process of the abstract query was the same in both cases. In this aspect, the difference is in the concrete query structure (SPARQL versus SQL) and not in the query target. The correctness of the results of the translation process was satisfactory, for Mondial and IMDB, in both environments, as compared with Coffman's benchmark. For Mondial, the tool correctly answered 33 queries, nearly 65%. For IMDB, 36 of 50 queries were correctly answered, approximately 72%.

Query build time. In all experiments, the query build time was nearly the same in both environments, since processing matches and constructing the abstract query were basically the same in both cases. In the relational environment, for the experiments with Mondial, the query build time accounted for 40-50% of the total elapsed time, on average; for the experiments with IMDB, it raised to slightly over 70-75%, possibly due to the ambiguity of IMDB data. By contrast, in the RDF environment, the query build time accounted for only 6-15% of the total elapsed time, on average. This behavior can be explained because matching is a costly process in both environments, but SPARQL queries take much longer to execute than SQL queries.

Total elapsed time. The total elapsed time was reasonable, on average, in all experiments. Even for a large database, such as IMDB, the total elapsed time was, on average, nearly 4 seconds, in the relational environment, but raised to 22 seconds, in the RDF environment. Indeed, the total elapsed time of the SQL queries was 4-6 times faster than the SPARQL queries, on average. Queries with contains filter use a text index, which is over all object values of the triples, for RDF datasets. But, for relational databases, there is a separate, smaller index for each text attribute. Thus, the elapsed time of SQL queries with a contains filter was smaller than that of SPARQL queries.

Navigation. We are not aware of any benchmark to evaluate this aspect, which is closely related to the users' interests. From the experiments (not detailed here), we may conclude that navigation through the results was much slower in the relational environment, since it involved several joins, as discussed in Section 3.2.

Navigation versus Querying. The previous observations suggest that the RDF environment should be favored when users frequently navigate over the keyword-based query results. Being reasonable in all experiments, the total elapsed time should not be an a priori argument to avoid the RDF environment.

5 Conclusions and Future Work

We described *QUIOW*, a tool designed to support keyword-based query processing for both RDF datasets with schemas and relational databases. Using full version of IMDB and the Mondial databases, the experiments indicated that the total elapsed time was

quite reasonable, on average, in both environments. Also, the experiments permitted us to conclude that the relational version reached better query performance, but had a poor navigation performance, when compared with the RDF version. Thus, if users tend to first query the data and then navigate through the results, the RDF version is an interesting alternative. The experiments suggest, as future work, to improve the performance of the keyword matching process by using alternative technologies, or by parallelization. We also plan to expand the tool to support other RDF stores and relational systems. Finally, we plan to explore users' preferences to deal with databases with very large schemas and use a domain ontology to expand keywords.

References

1. Aditya, B. et al. (2002). BANKS: Browsing and keyword searching in relational databases. VLDB 2002, 1083-1086.
2. Agrawal, S., Chaudhuri, S. & Das, G. (2002). DBXplorer: A system for keyword-based search over relational databases. ICDE 2002, 5-16.
3. Bast, H., Buchhold, B., & Haussmann, E. (2016). Semantic search on text and knowledge bases. Foundations and Trends® in Information Retrieval, 10(2-3), 119-271.
4. Bergamaschi, S. et al. (2016). Combining user and database perspective for solving keyword queries over relational databases. Inf. Syst. 55, C (Jan. 2016), 1-19.
5. Coffman, J., & Weaver, A. C. (2010). A framework for evaluating database keyword search strategies. CIKM 2010, 729-738.
6. Elbassuoni, S. & Blanco, R. (2011). Keyword search over RDF graphs. CIKM 2011, 237-242.
7. García, G.M., Izquierdo, Y.T., Menendez, E., Dartayre, F., & Casanova, M. A. (2017). RDF Keyword-based Query Technology Meets a Real-World Dataset. EDBT 2017, 656-667.
8. Gkirtzou, K., Papastefanatos, G. & Dalamagas, T. (2015). RDF Keyword Search based on Keywords-To-SPARQL Translation. NWSearch 2015, 3-5.
9. Han, S., Zou, L., Xu Yu, J. & Zhao, D. (2017). Keyword Search on RDF Graphs - A Query Graph Assembly Approach. arXiv:1704.00205 [cs.DB]
10. He, H., et al. (2007). Blinks: Ranked keyword searches on graphs. SIGMOD 2007, 305-316.
11. Hristidis, V. & Papakonstantinou, Y. (2002). DISCOVER: keyword search in relational databases. VLDB 2002, 670-681.
12. Le, W., Li, F., Kementsietsidis, A. & Duan, S. (2014). Scalable Keyword Search on Large RDF Data. IEEE TKDE Vol. 26, Issue 11 (Nov. 2014), 2774 – 2788.
13. Li, G., Ji, S., Li, C., & Feng, J. (2009). Efficient type-ahead search on relational data: a Tastier approach. SIGMOD 2009, 695-706.
14. Oliveira, P., Silva, A. & Moura, E. (2015). Ranking Candidate Networks of relations to improve keyword search over relational databases. ICDE 2015, 399-410
15. Tran, T., Wang, H., Rudolph, S. & Cimiano, P. (2009). Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. ICDE 2009, 405-416.
16. Yu, J., Qin, L. & Chang, L. (2009). Keyword Search in Databases. Morgan and Claypool.
17. Zenz, G., Zhou, X., Minack, E., Siberski, W. & Nejdl, W. (2009). 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), 166-176.
18. Zheng, W., Zou, L., Peng, W., Yan, X., Song, S. & Zhao, D. (2016). Semantic SPARQL similarity search over RDF knowledge graphs. Proc. VLDB Endowment 9(11), 840-851.
19. Zhou, Q. et al. (2007). SPARK: Adapting keyword query to semantic search. ISWC, 694-707.