Comparing Keyword-based Query Processing over RDF Datasets and Relational Databases

Yenier T. Izquierdo1,2, Grettel M. García1,2, Elisa S. Menéndez1,2, Marco A. Casanova1,2, 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 compares keyword-based query processing in two environments: RDF datasets with schemas and relational databases. The comparison is based on a tool that first translates a keyword-based query into an abstract query, and then compiles the abstract query into a SPARQL or a SQL query such that each result of the SPARQL or SQL 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 compare keyword-based query processing in two environments: RDF datasets with schemas and relational databases (which always have a schema). We base the comparison on the QUIOW tool, which uniformly implements keyword-based query processing for both environments. It evolved from an earlier implementation for RDF datasets, reported in [8]. QUIOW features an algorithm that first translates a keyword-based query into an abstract query, and then compiles the abstract query into a SPARQL or a SQL query such that each result of the query is an answer for the keyword-based query. The algorithm explores the schema to avoid user intervention during the translation process, and to minimize the number of joins in a query. The implementation is engineered to work with different RDF stores and relational DBMSs. The current implementation supports Oracle 12c, for both the RDF and relational environments, and PostgreSQL release 9.6, for the relational environment.
The experiments use RDF and relational versions of the Mondial database, compiled from geographical Web data sources, and of 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 [6].

The contributions of this paper can be summarized as follows. First, the paper unifies the notions of answer for keyword-based queries over the RDF and relational environments. Second, the paper describes a tool to support keyword-based queries for both RDF datasets and relational databases, provided that the RDF datasets have a schema. This is the first tool with these characteristics to be described in the literature, to the best of our knowledge. Finally, it includes extensive experiments to assess and compare keyword-based query processing in both the RDF and relational environments, and to analyze the challenges each environment poses.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 defines the basic concepts of keyword-based queries. Section 4 introduces the keyword translation algorithm the QUIOW tool uses. Section 5 describes extensive experiments to compare keyword-based query processing over RDF datasets and relational databases. Finally, Section 6 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 [3,20].

Early relational keyword-based query processing tools [1,2,11,12] 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 [4,16]. 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 [15] 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 [6] 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. In our comparison, we focus on the RDF schema-based tools out of fairness, since relational databases necessarily have a schema.

As a brief review some RDF schema-based approaches, Han et al. [10] described an algorithm that assembles a query from the keywords and experiments with DBpedia and Freebase. QUICK [21] 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. [9] 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 [23] 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. [18] combined the idea of generating summary graphs for the RDF graph, using the class hierarchy, to generate and rank candidate SPARQL queries. Le at al. [13] also proposed to process keyword queries using a summarization algorithm. Zheng et al. [22] also adopted a pattern-based approach. Elbassuoni and Blanco [7] described a technique to retrieve a set of subgraphs that match the keywords, and to rank them based on statistical language models.

Natural Language Question Answering is another related area. Systems in this category [14,17,9,19,24] accept natural language (NL) sentences as input, rather than keywords. A NL sentence conveys more semantics than a set of keywords, whereas a keyword-based query can be more concise and flexible. However, disambiguation and query intention understanding tend to be more difficult to address in keyword search.

The QUIOW tool, introduced in this paper, explores the (relational or RDF) schema, and is fully automatic. It is the first tool to be described in the literature, to the best of our knowledge, that supports keyword-based query processing for both the relational and RDF environments. The tool constructs a Steiner tree that covers a set of nodes (relation schemes or RDF classes) that match the largest set of keywords, and incorporates a backtracking step to further expand the keyword-query results by generating alternative (SQL or SPARQL) queries. The QUIOW tool also supports context-sensitive auto-completion.

The comparison presented in Section 5 adopts the keyword queries of Coffman’s benchmark, but it is based on relational and RDF implementations of the full versions of Mondial and IMDb, rather than the restricted versions of Coffman’s benchmark.

3 The Keyword Search Problem

3.1 RDF Environment

Recall that an RDF dataset $T$ is a set of RDF triples, and that $T$ induces a labeled graph $G_T$ such that the set of nodes of $G_T$ is the set of RDF terms that occur as subject or object of the triples in $T$ and there is an edge $(s,o)$ in $G_T$ labeled with $p$ iff the triple $(s,p,o)$ occurs in $T$.

Also, recall that an RDF schema is a set $S$ of RDF triples that use the RDF Schema 1.1 (RDF Schema or RDF-S) vocabulary [5] to declare classes, properties, property domains and ranges, and sub-class and sub-property axioms. Since an RDF schema is a set of RDF triples, it also induces a labelled graph.

In this paper, we assume that each RDF dataset $T$ follows an RDF schema $S$, with $S \subseteq T$, that is, the RDF schema is indeed defined and is part of the RDF dataset.

A keyword-based query $K$ is a set of literals, or keywords.
Let \( L \) be the set of all literals. Let \( \text{match}: L \times L \rightarrow [0,1] \) be a similarity function between literals such that \( \text{match}(s,t)=j \) indicates how similar \( s \) and \( t \) are: \( j=1 \) says that \( s \) and \( t \) are identical, and \( j=0 \) indicates that \( s \) and \( t \) are completely dissimilar. Finally, let \( \mu \in (0,1] \) be a similarity threshold. We leave \( \text{match} \) and \( \mu \) unspecified at this point.

A keyword \( k \in K \) has a metadata match with a triple \((r,p,v)\in S\) iff \( r \) is a class or property defined in \( S \) and \( \text{match}(k,v)\geq \mu \).

A keyword \( k \in K \) has a data match with a triple \((r,p,v)\in T\) iff \( \text{match}(k,v)\geq \mu \) (note that the triple \((r,p,v)\) must not be part of the RDF schema \( S \)).

We reinterpret the notion of answer for \( K \) over \( T \), introduced in [8], to separate metadata matches from data matches. This reinterpretation became necessary to extend it to relational databases, discussed in Section 3.2.

An answer for \( K \) over \( T \) is defined as a set \( A \) of triples in \( T \), partitioned into three sets, \( A_{CM}, A_{PM} \) and \( A_{DM} \), such that there are three possibly empty subsets of \( K \), denoted \( K/A_{CM}, K/A_{PM} \) and \( K/A_{DM} \), the keywords in \( K \) matched by \( A \), such that

1. For each \( k \in K/A_{CM} \), there is \((r,p,v)\in A_{CM}\) such that \( k \) has a metadata match with \((r,p,v)\) and \( r \) is declared as a class in \( S \).
2. For each \( k \in K/A_{PM} \), there is \((r,p,v)\in A_{PM}\) such that \( k \) has a metadata match with \((r,p,v)\) and \( r \) is declared as a property in \( S \).
3. For each \( k \in K/A_{DM} \), there is \((r,p,v)\in A_{DM}\) such that \( k \) has a data match with \((r,p,v)\).
4. For each \((r,p,v)\in A_{CM}\), there is \((s,\text{rdf:type},t)\in A_{DM}\) such that \( t=r \) or \( t \) is a subclass of \( r \) in \( S \).
5. For each \((r,p,v)\in A_{PM}\), there is \((s,t,v)\in A_{DM}\) such that \( t=r \) or \( t \) is a subproperty of \( r \) in \( S \).
6. \( G_{DM} \), the graph induced by \( A_{DM} \), is connected.
7. There is no other answer \( B \) for \( K \) over \( T \) such that \( B \) matches more keywords in \( K \) than \( A \).

We say that \( A_{CM} \cup A_{PM} \) is the metadata component of \( A \), and \( A_{DM} \) is the data component of \( A \). We define \( K/A = K/A_{CM} \cup K/A_{PM} \cup K/A_{DM} \) and say that \( A \) is total iff \( K/A = K \), and partial otherwise.

As expected, Conditions (1), (2) and (3) say that a keyword \( k \) may have a metadata match or a data match with a triple \((r,p,v)\) of the answer \( A \). Conditions (4) and (5) are not so obvious, though. They capture the interpretation that, if the user selects a class or a property (via a keyword), he actually wants an instance (and not all instances) of that class or property (other instances may be returned upon request). Condition (6) avoids disconnected answers. Condition (7) requires that an answer must match as many keywords in \( K \) as possible, since Conditions (1), (2) and (3) alone do not require that all keywords in \( K \) be matched in an answer.

This definition departs from [8] in that it requires that \( G_{DM} \) be connected. This is justified in so far as the translation algorithm introduced in [8], of which that described in Section 4 is an extension, outputs only connected answers. However, the definition allows \( G_{CM} \cup PM \), the graph induced by the metadata component \( A_{CM} \cup A_{PM} \) of \( A \), to be unconnected to account for matches between keywords and property descriptions.

The above definition does not force \( A \) to have a minimum set of triples, though. As in [8], to avoid this problem, we define a total order between answers, denoted \( \preceq \),
such that $A \leq B$ iff $|A| \leq |B|$, where $|\alpha|$ denotes the cardinality of a set $\alpha$. An answer $A$ for $K$ over $T$ is minimal iff there is no other answer $B$ for $K$ over $T$ such that $B \leq A$.

We finally define the problem of finding answers for keyword-based queries over RDF datasets (or, briefly, the RDF-KwS problem) as: “Given an RDF dataset $T$ and a keyword-based query $K$, find a possibly minimal answer for $K$ over $T$”.

### 3.2 Relational Environment

As usual, a relation scheme is denoted as $U[A_1,\ldots,A_n]$, where $U$ is the name and $A_1,\ldots,A_n$ are the attributes of the scheme. Sometimes we will refer to the relation scheme $U$, rather than to the relation scheme with name $U$. A foreign key is an expression of the form $F(U:L,V:M)$, where $F$ is the name of the foreign key, $U$ and $V$ are names of relation schemes, and $L$ and $M$ are lists of attributes of $U$ and $V$, respectively, with the same length. We say that $F(U:L,V:M)$ connects $U$ to $V$.

A relational schema is a pair $S=(\Sigma, \Phi)$, where $\Sigma$ is a set of relation schemes and $\Phi$ is a set of relational constraints, such that: (i) $\Phi$ has a unique primary key for each relation scheme in $\Sigma$; (ii) $\Phi$ has a mandatory attribute constraint for each attribute which is part of a key or primary key; (iii) if $\Phi$ has a foreign key of the form $F(U:L,V:M)$, then $\Phi$ also has a constraint indicating that $\Phi$ is the primary key of $V$. The relation schemes and their attributes may have descriptions, simply specified as text strings.

The schema $S=(\Sigma, \Phi)$ induces a labelled multigraph $G_S=(N_\Sigma, E_\Sigma, EL_\Sigma)$ such that $N_\Sigma=\Sigma$ and there is an arc $(U,V)\in E_\Sigma$, which $EL_\Sigma$ labels with $F$, iff there is a foreign key $F(U:L,V:M)$ in $\Phi$. Note that $G_S$ is a multigraph since there might be more than one foreign key between the same pair of schemes.

A consistent database state $\sigma$ of $S=(\Sigma, \Phi)$, or simply a database with schema $S$, is defined as usual and assigns a relation $\sigma[U]$ to each relation scheme $U$ in $\Sigma$ so that all constraints in $\Phi$ are satisfied.

A set $A$ of tuples from the relations in $\sigma$ induces a labelled multigraph $G_\sigma=(N_\Sigma, E_\Sigma, EL_\Sigma)$ such that $N_\Sigma=\Sigma$ and there is an arc $(u,v)\in E_\Sigma$, which $EL_\Sigma$ labels with $F$, iff $u\in \sigma[U]$ and $v\in \sigma[V]$, with $\sigma[L]=\sigma[M]$, and there is a foreign key $F(U:L,V:M)$ in $\Phi$.

Let $K$ be a keyword-based query. A keyword $k\in K$ has a metadata match with a relation scheme in $S$ with description $v$ iff $\text{match}(k,v)\geq \mu$. Likewise, $k\in K$ has a metadata match with an attribute $A$ of a relation scheme $U$ in $S$ with description $v$ iff $\text{match}(k,v)\geq \mu$.

A keyword $k\in K$ has a data match with $r[A]$, where $U$ is a relation scheme in $S$, $A$ is an attribute of $U$, and $t\in \sigma[U]$, iff $\text{match}(k,t[A])\geq \mu$.

An answer for $K$ over a database $\sigma$, with relation schema $S$, is a triple $A=(A_{SM},A_{AM},A_{TM})$, where:

- $A_{SM}$ is a set of relation scheme of $S$
- $A_{AM}$ is a set of pairs $(U,A)$, where $U$ is a relation scheme of $S$ and $A$ is an attribute of $U$
- $A_{TM}$ is a set of triples $(U,A,t)$, where $U$ is a schema name of $S$, $A$ is an attribute of $U$, and $t\in \sigma[U]$

such that there are three possibly empty subsets of $K$, denoted $K/A_{SM}$, $K/A_{AM}$ and $K/A_{TM}$, the keywords in $K$ matched by $A$, such that:
1. For each \( k \in K/A_{SM} \), there is \( U \in A_{SM} \) such that there is a metadata match between \( k \) and the relation scheme \( U \).
2. For each \( k \in K/A_{SM} \), there is \( (U,A) \in A_{SM} \) such that there is a metadata match between \( k \) and attribute \( A \) of \( U \).
3. For each \( k \in K/A_{TM} \), there is \( (U,A,t) \in A_{TM} \) such that there is a data match between \( k \) and \( t[A] \).
4. For each \( U \in A_{SM} \), there is \( (U,A,t) \in A_{TM} \).
5. For each \( (U,A) \in A_{SM} \), there is \( (U,A,t) \in A_{TM} \).
6. \( G_{TM} \), the multigraph induced by the tuples in \( A_{TM} \), is connected.
7. There is no other answer \( B \) for \( K \) over \( T \) such that \( B \) matches more keywords in \( K \) than \( A \).

We say that \( A_{SM} \) and \( A_{AM} \) are the metadata components and \( A_{TM} \) is the data component of the answer \( A \). We define \( K/A = K/A_{SM} \cup K/A_{AM} \cup K/A_{TM} \).

As much as possible, these conditions are a parallel of those imposed on the notion of answer for RDF datasets. The major difference comes from the fact that an RDF schema is also a set of triples, just like an RDF dataset, whereas a relational schema is obviously quite different from a relational database state. Therefore, metadata matches and data matches are treated differently in the relational case.

We again define a total order between answers “\( \preceq \)” such that \( A \preceq B \) iff \( |A_{SM} \cup A_{AM} \cup A_{TM}| \leq |B_{SM} \cup B_{AM} \cup B_{TM}| \), where \( A = (A_{SM}, A_{SM}, A_{TM}) \) and \( B = (B_{SM}, B_{AM}, B_{TM}) \). An answer \( A \) for \( K \) over a database \( \sigma \), with relation scheme \( S \), is minimal iff there is no other answer \( B \) for \( K \) over \( \sigma \) such that \( B \preceq A \).

Finally, we define the problem of finding answers for keyword-based queries over relational databases (or, briefly, the R-KwS problem) as: “Given a relational database \( \sigma \) and a keyword-based query \( K \), find a possibly minimal answer for \( K \) over \( \sigma \).”

### 4 The Keyword Search Tool

This section describes the QUIOW tool, which uniformly implements keyword-based query processing for the RDF and relational environments, and is fully operational. That is, QUIOW was designed to address both the RDF-KwS and the R-KwS problems.

QUIOW features an algorithm that first translates a keyword-based query into an abstract query, and then compiles the abstract query into a SPARQL or a SQL query such that each result of the query is an answer for the keyword-based query.

#### 4.1 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 scheme \( S \) 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.

Let us now turn to 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 result of the matching process 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 [8].

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 (by a reduction to the bin-packing 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.

### 4.2 Examples

This section illustrates how the keyword translation algorithm works in both RDF and relational environments, through an example using Mondial, whose ER-Diagram is available at [https://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-ER.pdf](https://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-ER.pdf).

Figure 1 illustrates the translation algorithm. Figure 1(a) shows the keyword-based query \( K = \{ \text{Country, Province, Desert, Atacama} \} \). 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 \( \text{Country}, \text{Province}, \text{Desert}, \) and \( \text{Geo\_Desert} \); the first three classes correspond to nucleuses and the fourth node just completes the tree.
- The arcs represent object properties between such classes.
- The classes that correspond to nucleuses reflect the following matches:
  - The keywords \( \text{Country}, \text{Province}, \) and \( \text{Desert} \) have an exact metadata match with the labels of classes \( \text{Country}, \text{Province} \) and \( \text{Desert} \).
  - The keyword \( \text{Atacama} \) matches a value of property \( \text{Label} \) whose domain is the class \( \text{Desert} \).

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 \( \text{Country}, \text{Desert}, \) and \( \text{Province} \), 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 \( \text{Atacama} \).
- Lines 11 to 13 translate instances to their labels, which are user-friendly.

The next subsection discusses the use of labels and SELECT SPARQL queries.

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

- Lines 1 to 7 capture the primary keys (which have multiple attributes) and the \( \text{META\_REPCOL} \) columns of the relation schemes.
- Line 8 corresponds to the nodes of the Steiner tree.
- Line 10 represents the match with the keyword \( \text{Atacama} \).
- Lines 11 to 13 represent equijoins that correspond to the arcs of the Steiner tree.

Note: for each relation, the data preparation stage adds a new column, called \( \text{META\_REPCOL} \), whose values correspond to the instance labels of the RDF version.
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, as defined in Section 3. 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. \(\text{SELECT} \ ?\text{Class} \ ?\text{Property} \ ?\text{Value} \ ?\text{Label} \)
   2. \(\text{WHERE}\{\)
   3. \(\text{?uri} \ \text{rdfs:label} \ ?\text{Class} \)
   4. \(\text{?uri} \ ?\text{prop} \ ?\text{Value} \)
   5. \(\text{OPTIONAL}\{\ ?\text{value} \ \text{rdfs:label} \ ?\text{Label}\} \)
   6. \(\text{?uri} \ \text{rdf:type} \ ?\text{c} \)
   7. \(\text{?c} \ \text{rdfs:label} \ ?\text{Class}\)

2) SPARQL query \(Q_2\):
   1. \(\text{SELECT} \ ?\text{Class} \ ?\text{Value} \ ?\text{Label} \)
   2. \(\text{WHERE}\{\)
   3. \(\text{?uri} \ ?\text{prop} \ ?\text{Label} \)
   4. \(\text{OPTIONAL}\{\ ?\text{Value} \ \text{rdfs:label} \ ?\text{Label}\} \)
   5. \(\text{?Value} \ \text{rdf:type} \ ?\text{c} \)
   6. \(\text{?c} \ \text{rdfs:label} \ ?\text{Class}\)

Recall from Section 4 that all classes and properties have a mandatory label. Consider query \(Q_1\), and assume that variable \(?\text{uri}\) 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:
   1. SELECT pk₁, ..., pkₙ, Label, p₁, p₂, ..., pₙ
   2. FROM T
   3. WHERE <instance filter>

   that queries table T (Line 2) to retrieve the primary keys, instance label, and properties (Line 1), adding to the WHERE clause the filter defined by the identifier created for the target instance (Line 3). In our example, the instance filter is built using the primary key values for instance Antofagasta.

2. Compute a set $J_F$ of tables such that $V \in J_F$ iff $T$ has a foreign key to $V$. For each $V \in J_F$, 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 $J_T$ of tables such that $U \in J_T$ iff $U$ has a foreign key to $T$. For each table $U \in J_T$, 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 $J_F \cup J_T$.

   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 3 times faster than in the relational database.

5 Experiments

5.1 Experimental Setup

Environment. All experiments used the QUIOW tool, implemented as a RESTful Web application developed in Java. For the Mondial experiments, the app ran on a desktop with OS Windows 7 Ultimate, a quad-core processor Intel(R) Core(TM) i5-2450M

---

Fig. 2. Sample query results and navigation panel.

<table>
<thead>
<tr>
<th>Country</th>
<th>Desert</th>
<th>Province</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chile</td>
<td>Atacama</td>
<td>Antofagasta</td>
</tr>
<tr>
<td>Chile</td>
<td>Atacama</td>
<td>Arica y Parina</td>
</tr>
<tr>
<td>Chile</td>
<td>Atacama</td>
<td>Atacama</td>
</tr>
<tr>
<td>Chile</td>
<td>Atacama</td>
<td>Tarapacá</td>
</tr>
</tbody>
</table>

(a) Results for the keyword-based query $K = \{\text{Country, Province, Desert, Atacama}\}$.

(b) Navigation panel that results from clicking on the instance Antofagasta.
CPU @ 2.50GHz, 4 GB of RAM. The relational database and RDF dataset were stored in Oracle 12c, running on a quad-core machine with processor Intel(R) Core(TM) i5 CPU 660 @ 3.33GHz, 7GB of RAM, and 4,096 KB of cache size.

For the IMDb experiments, the app ran on a desktop with OS macOS Sierra, a 1.7 GHz Intel Core i5, 4 GB of RAM. Oracle 12c was also used, running on a 2x deca-core Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz, 128GB RAM, 32KB Cache L1.

Data setup. We selected the relational version 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 Coffman’s benchmark [6].

We used the MAPGEN tool to generate RDF versions of these relational databases. MAPGEN materializes an RDF dataset by triplifying a relational database via a set of relational-to-RDF mappings, and creates indexes to support keyword search.

Table 1 shows basic statistics about the RDF datasets and relational databases used in the experiments.

Table 1. Statistics – Mondial and IMDb Datasets.

<table>
<thead>
<tr>
<th>RDF Dataset</th>
<th>#Triples</th>
<th>Relational Database</th>
<th>#Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mondial</td>
<td>IMDb</td>
<td></td>
</tr>
<tr>
<td>Class declarations</td>
<td>40</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Object property declarations</td>
<td>62</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>Datatype prop. declarations</td>
<td>130</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>Indexed properties</td>
<td>71</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Indexed prop. instances</td>
<td>11,094</td>
<td>12,609,418</td>
<td></td>
</tr>
<tr>
<td>Class instances</td>
<td>43,869</td>
<td>70,520,744</td>
<td></td>
</tr>
<tr>
<td>Object property instances</td>
<td>63,652</td>
<td>204,917,673</td>
<td></td>
</tr>
<tr>
<td><strong>Total number of triples</strong></td>
<td><strong>235,387</strong></td>
<td><strong>382,295,213</strong></td>
<td><strong>0.11 60.63</strong></td>
</tr>
</tbody>
</table>

Measurements. We measured two basic variables: query build time, which is the time taken to process matches and construct the SQL or SPARQL query; and total elapsed time, which is time from the submission of the keyword-based query until the display of the first 75 results.

5.2 Experiments with Mondial

We executed the keyword-based queries defined in Coffman’s benchmark [6] against the Mondial relational database and its triplified version. For both versions, Figure 3 shows, on the Y-axis, the query build time and the total elapsed time, in seconds, of each query in Coffman’s 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 much 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). Section 5.4 will further detail these points.
5.3 Experiments with IMDb

Contrasting with the Restricted IMDb, the Full IMDb features a much more complex conceptual schema (see Table 1). Furthermore, while the Restricted IMDb has data only about movies, the Full IMDb has data about movies, series, episodes, video games, etc. We ran the keyword-based queries defined in Coffman’s benchmark [6] against the Full IMDb relational database and its triplified version.

In order to reduce ambiguity when using the Full IMDb, as compared with the Restricted 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, movie, TV series, 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 [8, Section 4.1], this affects the computation of the scores.

Again, for the SPARQL and SQL versions, Figure 4 shows, on the Y-axis, the query build time and the total elapsed time, in seconds, of each query in Coffman’s 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 again much larger than the SQL total elapsed time (shown as a cross), except for a few queries (dots near crosses); 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). We note that the use of quotes only improved the query build time, and did not change the final results. Section 5.4 will again detail these points.

5.4 Lessons Learned

This section summarizes the lessons learned by comparing keyword-based query processing over the RDF and relational environments, considering the following aspects: preparation and maintenance, quality of the query results, query build time, total elapsed time, and navigation over the query 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>
<thead>
<tr>
<th>Table 2. Summary of the experiments.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mondial</td>
</tr>
<tr>
<td>Avg.</td>
</tr>
<tr>
<td>(in seconds)</td>
</tr>
<tr>
<td>Relational</td>
</tr>
<tr>
<td>RDF</td>
</tr>
<tr>
<td>Max.</td>
</tr>
<tr>
<td>RDF</td>
</tr>
<tr>
<td>Min.</td>
</tr>
<tr>
<td>RDF</td>
</tr>
</tbody>
</table>

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 of Mondial. 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. A detailed discussion about the correctness of the translation process is out of the scope of this paper, but we remark that the results were satisfactory, for both Mondial and IMDb, in both environments, as compared with Coffman’s benchmark.

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 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 6 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 total 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 4.2. In fact, navigation in the relational case will have a performance similar to the RDF case only if the relational schema is simple, or at least has a small set of foreign keys, which implies a small number of joins to be executed during navigation.

**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.

## 6 Conclusions and Future Work

We first described the QUIOW tool, designed to support keyword-based query processing both for RDF datasets with schemas and relational databases. With the help of QUIOW, we were able to compare keyword-based query processing in the RDF and relational environments. Using a full version of IMDb and the Mondial database, the experiments indicated that the total elapsed time was quite reasonable, on average, in both environments. The experiments also permitted us to conclude that the relational version reached better query performance, but had a poorer 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.

**Acknowledgments.** This work was partly funded by CNPq grants 303332/2013-1 and 140920/2017-0, by CAPES/DSE grant 88881.134081/2016-01.
References


