Industrial and Applications Paper

RDF Keyword-based Query Technology Meets a Real-World Dataset

Grettel M. García[1,2], Yenier T. Izquierdo[1,2], Elisa S. Menendez[1,2], Frederic Dartayre[1], Marco A. Casanova[1,2]
[1]Instituto TecGraf – Pontifícia Universidade Católica do Rio de Janeiro
[2]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225 – Rio de Janeiro, RJ – Brazil CEP 22451-900
+55-21-3527-1500
ggarcia@inf.puc-rio.br, yizquierdo@inf.puc-rio.br, emenendez@inf.puc-rio.br, fdartayre@tecgraf.puc-rio.br, casanova@inf.puc-rio.br

ABSTRACT
This paper presents the results of an industrial project, conducted by the TecGraf Institute and Petrobras (the Brazilian Petroleum Company), to develop a tool to facilitate access to a large database, with hydrocarbon exploration data, by combining RDF technology with keyword search. The tool features an algorithm to translate a keyword query into a SPARQL query such that each result of the SPARQL query is an answer for the keyword query. The algorithm explores the RDF schema of the RDF dataset to generate the SPARQL query and to avoid user intervention during the translation process. The tool offers an interface which allows the user to specify keywords, as well as filters and unit measures, and presents the results with the help of a table and a graph. Finally, the paper describes experiments which show that the tool achieves very good performance for the real-world industrial dataset and meets users' expectations. The tool was further validated against full versions of the IMDb and Mondial datasets.

CCS Concepts
Information systems → Information retrieval → Information retrieval query processing → Query reformulation

Keywords
Keyword search; SPARQL; RDF.

1. INTRODUCTION
Keyword search is typically associated with information retrieval systems, especially those designed for the Web. The user just 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 list of keywords. These systems also usually offer an advanced search interface, which the user may take advantage to specify Boolean expressions involving the keywords, or limit the search to a subset of the documents, such as an Internet domain. Information retrieval systems typically implement algorithms to rank the results of a keyword search so that, hopefully, the user will find the most interesting documents at the top of the list. The success of such systems may therefore be credited to: (1) a very simple user interface; (2) an efficient document retrieval mechanism; and (3) a ranking algorithm which meets user expectations.

By contrast, database management systems offer sophisticated query languages to access structured data. It is up to the database applications to create user interfaces that hide the complexity of the query language. User interfaces are often designed as a stack of pages with numerous "boxes" that the user must fill with his search parameters. This traditional design may end up with uncomfortable user interfaces, which are amply justified, though, when the user has to specify exact data, such as a flight number and a flight date.

Hitting the middle ground, we find database applications that offer keyword search interfaces over conventional databases or, in short, *keyword search database applications*. These applications should reach a performance similar to information retrieval applications, despite the fact that the underlying data is stored in a conventional database. Furthermore, they should free the user from filling "boxes" with exact data by compiling keyword searches to the query language supported and by ranking the results in a meaningful way from the user point of view.

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. The adoption of RDF as the underlying data model has some interesting advantages. The most obvious is the flexibility RDF offers by modeling data as RDF triples of the form (s,p,o), which asserts that resource s has property p with value o. Of special interest for keyword search is the fact that RDF imposes no strict distinction between data and metadata, that is, a keyword may match the name or description of a class or of a property in the same way as it matches a data value. RDF management systems also sometimes offer an inference layer so that one may expand the stored RDF data with derived data in ways that surpass (relational) views. Thus, a keyword may match derived data as much as stored data. Lastly, an RDF dataset may be treated as a graph, which allows the use of graph concepts and algorithms to explore the data.

The paper summarizes the results of an industrial project, conducted by the TecGraf Institute and Petrobras (the Brazilian Petroleum Company), to facilitate access to a large relational database, with hydrocarbon exploration data, by combining RDF technology with keyword search. The prototype application is being deployed to the production environment to be tested on a large scaled by the target users.

The contributions of this paper are as follows. First, the paper defines the concept of an answer for a keyword-based query over an RDF dataset. Second, the paper introduces an algorithm to translate keyword-based queries to SPARQL queries that takes

advantage of the schema of the RDF dataset to avoid user intervention and achieve good performance, even for large RDF datasets. Third, it describes an interface that allows the user to specify keywords with the help of an auto-completion feature, as well as filters and unit measures, such as "wells with depth between 1,000m and 2,000m". The interface presents an answer to the user in a way that reduces the cognitive overhead required to navigate through an RDF graph. Lastly, the paper describes experiments that show that the tool achieves very good performance for the real-world industrial dataset and meets users' expectations. The tool was further validated against full versions of the IMDb and Mondial datasets.

The remainder of this paper is organized as follows. Section 2 summarizes related work. Section 3 provides a brief background on RDF and defines the basic concepts of keyword-based queries. Section 4 covers the translation algorithm and the user interface. Section 5 describes experiments to assess the performance and usability of the tool. Finally, Section 6 contains the conclusions.

2. Related Work

Keyword-based query processing. Tools that implement keyword-based queries over relational databases and RDF datasets have been investigated for some time. We may distinguish between tools that are *schema-based*, in the sense that they use information about the conceptual schema to compile a keyword-based query into an SQL or SPARQL query, from those that are *graph-based*, in the sense that operate directly on the data. We may also identify *pattern-based* tools, which hit the middle ground, in the sense that they mine patterns from the RDF dataset to be used in lieu of the conceptual schema. It is also useful to distinguish between *fully automatic* tools from tools that resort to user intervention during the processing of the keyword-based queries.

BANKS [1] and BLINKS [11] are examples of early relational graph-based tools. Relational schema-based tools explore the foreign 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 first proposed in DISCOVER [12] and DBXplorer [2] and adopted in a quite a few tools, including recent ones [15].

SPARK [28] offers an example of an early pattern-based RDF graph-based tool. Tran et al. [21] combine the idea of generating summary graphs for the original RDF graph, using the class hierarchy, to generate and rank candidate SPARQL queries. Zhang et al. [26] investigated a solution to this problem, backed up by experiments over a subset of the original IMDb, a selection of articles from Wikipedia, and the Mondial dataset. More recently, Yang et al. [24] proposed to mine tree patterns that will then connect together the keywords specified by the user; the tree patterns are ordered by relevance using their size, the pagerank of the nodes and the quality of keyword match. Zheng et al. [27] proposed a systematic method to mine semantically equivalent structure patterns to summarize the knowledge graph and, thereby, circumvent the lack of an RDF schema. Finally, De Virgilio [7] proposed an RDF keyword-based query processing strategy based on tensor calculus, later extended to a distributed environment [8].

QUICK (*QUery Intent Constructor for Keywords*) [25] is an RDF schema-based tool designed to translate keyword-based queries to SPARQL queries with the help of the users, who choose a set of intermediate queries, that the tool ranks and executes.

The tool described in this paper is schema-based and fully automatic. We borrowed from the early relational graph-based tools the idea of minimizing the number of equijoins by generating a Steiner tree of a graph induced by the RDF schema. However, we introduce the (new) concept of a *nucleus*, consisting of a class, a list of properties, and a list of property values, which is in some sense analogous to a tuple and helps translate keyword-based queries to SPARQL queries. The Steiner tree will then connect the classes of the nucleus that cover the keywords.

QUICK is the tool closest to ours in so far as both tools explore the RDF schema to synthesize SPARQL queries. However, differently from QUICK, we opted for a fully automatic translation. This was possible essentially because our tool was designed to operate over an RDF dataset which has a rich schema and whose data exhibits low ambiguity.

Triplification of the relational database. Triplification, the process of mapping a relational database to an RDF dataset, is based on well-established technologies, backed up by a standardized mapping language, R2RML [6]. However, relational databases are usually normalized and, therefore, should not be directly mapped to RDF. To deal with this issue, we followed the strategy proposed in [22], which suggests to first create relational views that define an unnormalized relational schema and then write the R2RML mappings on top of these views.

In fact, the judicious design of the RDF schema helps the translation process from keyword-based queries to SPARQL queries. This requires additional comments. First, the assumption that the RDF dataset has a known schema should not be viewed as a demerit. Indeed, a large fraction of the LOD datasets do have a known schema (vocabulary or ontology) [17]. Furthermore, in a corporate environment, such as ours, RDF datasets are frequently triplifications of relational databases. Second, even when one cannot change the (relational or RDF) schema, one may add a conceptual layer, defined with the help of views, that hide normalizations, in the relational case, or poorly designed RDF schemas, which in both cases would lead to ambiguities when processing keyword-based queries.

Benchmarks. Coffman and Weaver [4] describe a benchmark which uses a simplified, relational version of IMDb, a subset of Wikipedia, and a subset of the Mondial dataset. The keyword queries are mostly very simple.

Guo et al. [9] introduced LUBM, a benchmark for OWL knowledge base systems, which consists of an ontology for the university domain, synthetic OWL data scalable to an arbitrary size, and 14 SPARQL queries. More recently, an ontology-based data access benchmark, the *NPD Benchmark* [13][20], was constructed using real data from the Norwegian Petroleum Directorate (NPD) FactPages. The benchmark generates, from the NPD data, datasets of increasing size; the SPARQL queries were formulated by domain experts from an informal set of questions provided by regular users of the FactPages. Finally, Qiao and Özsoyoğlu [18] published the RBench, an application-specific RDF benchmarking tool that takes an RDF dataset from any application as a template, and generates a set of synthetic datasets and different types of queries systematically.

Although our tool was designed for a specific RDF dataset, we decided to test it against other datasets. However, a direct comparison with other keyword search tools turned out to be problematic, for two basic reasons. First, contrasting with Coffman's benchmark setting, our tool takes advantage of more complex RDF schemas and of keyword-based queries with a fairly large number of keywords – the query profile of our typical users – to avoid user intervention during the synthesis of the SPARQL query. Second, our tool presently does not incorporate reasoning features, i.e., we deal with a standard dataset and not with a

knowledge base. In the end, we opted to further test our tool against the full versions of IMDb and Mondial, which feature conceptual schemas with a complexity closer to the schema of the target industrial dataset. We used the same list of keyword queries as in Coffman's benchmark, albeit they are much simpler than those expected from our typical users, as already pointed out.

3. BASIC DEFINITIONS

3.1 RDF Essentials

In this section, we summarize some basic concepts pertaining to the *Resource Description Framework* (RDF) [5]. The reader familiar with RDF may skip this section.

An *Internationalized Resource Identifier* (*IRI*) is a global identifier that denotes a resource. We will use the terms IRI and resource interchangeably. A *literal* is a basic value, such a string, a number, or a date. A *blank node* acts as a local identifier; a blank node can always be replaced by a new, globally unique IRI (a *Skolem IRI*). An *RDF term* is either an IRI, a blank node or a literal. The sets of IRIs, blank nodes and literals are disjoint. In the rest of this paper, ***IRI*** denotes the set of all IRIs and ***L*** the set of all literals.

RDF models data as triples of the form *(s,p,o)*, where *s* is the *subject*, *p* is the *predicate* and *o* is the *object* of the triple. An RDF triple *(s,p,o)* says that some relationship, indicated by *p*, holds between the subject *s* and object *o*. The subject of a triple is an IRI or a blank node, the predicate is an IRI, and the object is an IRI, a blank node or a literal.

A set *T* of RDF triples, or an *RDF dataset*, is equivalent to 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*. Therefore, we will use the concepts of RDF dataset and RDF dataset graph interchangeably. Note that an IRI may occur both as a node and as an edge label in the same graph.

RDF offers enormous flexibility but, apart from the rdf:type property, which has a predefined semantics, it provides no means for defining application-specific classes and properties. Instead, such classes and properties, and hierarchies thereof, are described using extensions to RDF provided by the *RDF Schema 1.1* (RDF Schema or RDF-S) [3]. In RDF-S, a *class* is any resource having an rdf:type property whose value is the qualified name rdfs:Class of the RDF Schema vocabulary. Likewise, a *property* is any resource having an rdf:type property whose value is the qualified name rdfs:Property. The rdfs:domain property is used to indicate that a given property applies to a designated class, and the rdfs:range property is used to indicate that the values of a particular property are instances of a designated class or, alternatively, are instances (i.e., literals) of an XML Schema datatype. RDF-S also offers the rdfs:subClassOf and the rdfs:subPropertyOf properties that allow the specification of sub-class and sub-property axioms. Finally, RDF-S features a property, rdfs:comment, used to associate a comment with a resource, and a property, rdfs:label, used to assign a name to a resource.

An *RDF schema* is a set *S* of RDF triples that use the RDF-S vocabulary to declare classes, properties, property domains and ranges, and sub-class and sub-property axioms. Viewed as a set of RDF triples, *S* is also equivalent to a labelled graph G_S.

A *simple RDF schema* is a RDF schema that contains only class declarations, object and datatype property declarations and sub-class axioms (and no sub-property axioms). We then introduce a labelled graph, D_S, called an *RDF schema diagram*, defined as follows: (1) the nodes of D_S are the classes declared in *S*; and (2) there is an edge from class *c* to class *d* labelled with *subClassOf* iff *c* is declared as a subclass of *d* in *S*, and there is an edge from class *c* to class *d* labelled with *p* iff *p* is declared in *S* as an object property with domain *c* and range *d*.

Very briefly, we say that an RDF dataset *T follows* an RDF schema *S* iff we have: (1) $S \subseteq T$; (2) all classes and properties used in *T*, except those in *S* itself, are declared in *S*; and (3) the triples in *T*, again except those in *S*, satisfy all restrictions imposed by the declarations in *S* [3]. Note that, by this definition, the RDF schema is contained in the RDF dataset, which is convenient for our purposes (see Section 3.2).

Finally, SPARQL is a query language specifically designed to access RDF datasets [10]. SPARQL offers two types of queries. A *SELECT query* returns tabular data, whereas a *CONSTRUCT query* returns a set of RDF triples. The body of a SPARQL query is a list of *triple patterns*, defined like RDF triples, except that the subject, predicate or object can be a variable. The evaluation of a SPARQL query *Q* against an RDF dataset *T* binds values to the variables using a *solution mapping* σ in such a way that the WHERE clause of *Q* generates a subgraph of *T*. An *answer* of *Q* is an instantiation of the variables in the target clause of *Q* generated by σ.

3.2 Keyword-Based Queries

Let *T* be an RDF dataset and G_T be the corresponding RDF graph. We assume that *T* follows an RDF schema *S*, with $S \subseteq T$.

A *keyword-based query K* is simply a set of literals, or *keywords*.

Recall that ***L*** is the set of all literals. Let *match*: $\boldsymbol{L}\times\boldsymbol{L} \rightarrow [0,1]$ be a similarity function between literals such that $match(s,t)=j$ 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. We also introduce a *similarity threshold* $\sigma\in(0,1]$. We leave *match* and σ unspecified at this point.

The set ***MM[K,S]*** of *metadata matches* between *K* and the metadata descriptions of the classes and properties in *S* is defined as:

$$\boldsymbol{MM}[K,T] = \{\ (k,(r,p,v))\in K\times T \ /\ (r,p,v)\in S \wedge match(k,v)\geq\sigma\ \}$$

The set ***VM[K,T]*** of *property value matches* between *K* and property values of *T* is defined as (recall that $S \subseteq T$):

$$\boldsymbol{VM}[K,T] = \{\ (k,(r,p,v))\in K\times T \ /\ (r,p,v)\notin S \wedge match(k,v)\geq\sigma\ \}$$

The set of *matches* between *K* and *T* is then defined as:

$$\boldsymbol{M}[K,T] = \boldsymbol{MM}[K,T] \cup \boldsymbol{VM}[K,T]$$

An *answer* for *K* over *T* is a subset *A* of *T* such that:

(1) There is a subset of *K*, denoted *K/A*, such that, for each $k\in K/A$:
   a. There are $(s,\text{rdf:type},c_n)$, $(c_n,\text{rdfs:subClassOf},c_{n-1})$,..., $(c_1,\text{rdfs:subClassOf},c_0)$ and (c_0,p_0,v_0) in *A* such that $(k,(c_0,p_0,v_0))\in\boldsymbol{MM}[K,T]$; or
   b. There are (s,q_n,v_n), $(q_n,\text{rdfs:subPropertyOf},q_{n-1})$,..., $(q_1,\text{rdfs:subPropertyOf},q_0)$ and (q_0,p_0,v_0) in *A* such that $(k,(q_0,p_0,v_0))\in\boldsymbol{MM}[K,T]$; or
   c. There is $(r,p,v)\in A$ such that $(k,(r,p,v))\in\boldsymbol{VM}[K,T]$.

(2) There is no other answer *B* for *K* over *T* such that $K/A \subset K/B$.

We say that *K/A* is the set of keywords *matched* by *A*.

Condition (1a) says that a keyword *k* has a class metadata match for a class c_0 and the answer *A* must contain an instance of c_0 or one of its sub-classes c_n, in which case *A* must include all triples indicating that c_n is a sub-class of c_0. Likewise, Condition (1b) says that a keyword *k* has a property metadata match for a property q_0 and the answer *A* must contain an instance of q_0 or one of its sub-properties

q_n, in which case A must include all triples indicating that q_n is a sub-property of q_0. Condition (1c) simply says that k matches the literal of a triple (r,p,v) in A. Also, Condition (1) does not require that all keywords in K be matched in an answer. Indeed, we say that A is *total* iff $K/A = K$, and *partial* otherwise. Condition (2) requires that an answer must match as many keywords in K as possible.

The definition of an answer is quite liberal. In particular, it allows an answer A to be a set of disconnected triples, as in Figure 1c. To circumvent this problem, we define a partial order between answers as follows. Given a directed graph G, let $|G|$ denote the number of nodes and edges of G and $\#c(G)$ denote the number of connected components of G, when the direction of the edges of G is disregarded. We define a partial order "<" for graphs such that, given two graphs G and G',

$$G < G' \text{ iff } (\#c(G) + |G|) < (\#c(G') + |G'|) \text{ or } (\#c(G) + |G|) = (\#c(G') + |G'|) \text{ and } \#c(G) < \#c(G')$$

We use the partial order "<" between graphs to compare answers. We say that an answer A *is smaller than* an answer B iff $G_A < G_B$, where G_A and G_B are the RDF graphs of A and B (which may include metadata, since the RDF schema is part of the dataset). An answer A for K over T is *minimal* iff there is no other answer B for K over T such that $G_A < G_B$.

In this paper, we focus on heuristics to find, possibly, minimal answers for keyword-based queries. We are especially interested in heuristics that, given a keyword-based query K, generate a CONSTRUCT SPARQL query Q over T which is a *correct query interpretation* for K, in the sense that each set of triples returned by Q is an answer for K over T and, preferably, a minimal answer.

Example 1: Consider an RDF dataset T, whose RDF graph G_T is shown in Figure 1a, where the darker boxes with boldface italic labels partly denote the RDF schema. Consider the keyword-based query K = {`Mature`, `Sergipe`}. Then, we have the following set of matches of K for T:

$$\begin{aligned} \boldsymbol{M}[K,T] = \{ & (\texttt{Mature}, (r_1, \text{:stage}, \text{"Mature"})), \\ & (\texttt{Mature}, (r_2, \text{:stage}, \text{"Mature"})), \\ & (\texttt{Sergipe}, (r_1, \text{:inState}, \text{"Sergipe"})), \\ & (\texttt{Sergipe}, (r_3, \text{:name}, \text{"Sergipe Field"})) \} \end{aligned}$$

There are several possible answers for K over T, two of which are represented in Figures 1b and 1c (in the form of their RDF graphs; note that the dashed node labelled "Alagoas" is not part of answer A_2; it is depicted just to alert that A_2 includes resource r_2). Note that answers A_1 and A_2 match both keywords in K. However, since $|G_{A1}|$=5, $|G_{A2}|$=6, #c(G_{A1})=1, and #c(G_{A2})=2, we have $G_{A1} < G_{A2}$, and hence A_1 should be preferred to A_2.

However, the keyword-based query K is ambiguous, since it does not indicate whether the keyword `Sergipe` refers to a state or to an oil field. To disambiguate, we might consider the keyword-based query K' = { `Mature`, `"located in"`, `"Sergipe Field"`}. Indeed, we would obtain answer A_3, shown in Figure 1d. The dashed rectangle highlights components of the RDF schema which are part of the answer. Indeed, note that there is a property metadata match between the keyword "`located in`" and the label value "located in" of property :locIn (note again that the dashed node labelled "Alagoas" is not part of the answer). Furthermore, as required by the definition of an answer, note that (r_2, :locIn, r_3) is an instance of property :locIn. Naturally, a second answer to K', similarly defined but involving resource r_1, would also be acceptable, since r_1 represents a mature well and is located in the Sergipe Field.

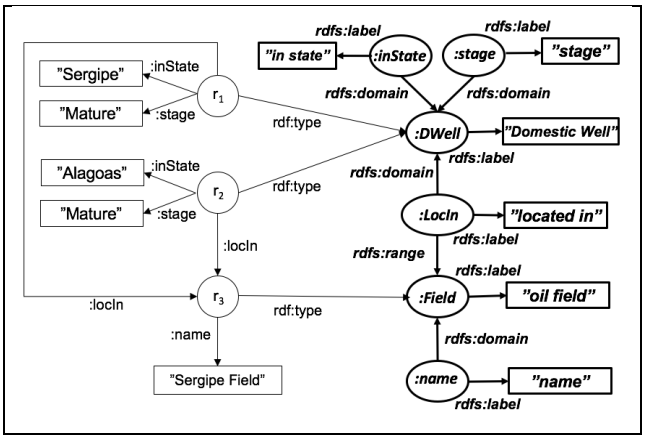


Figure 1a. The RDF graph G of Example 1.

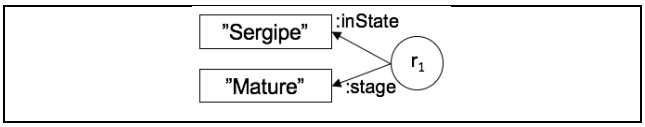


Figure 1b. An answer A_1 for the keyword-based query K = {`Mature`, `Sergipe`}.

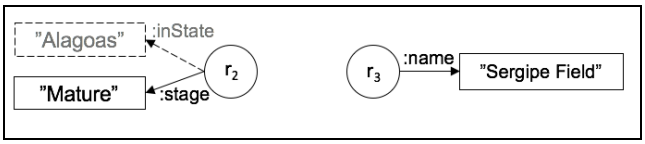


Figure 1c. A second answer A_2 for the keyword-based query K = {`Mature`, `Sergipe`}.

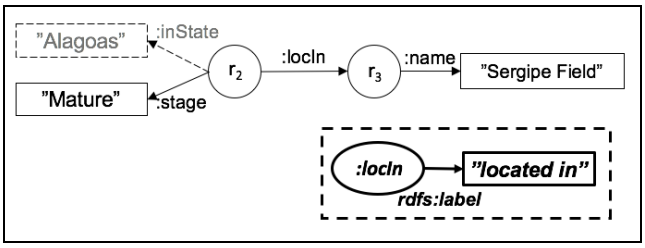


Figure 1d. An answer A_3 for the keyword-based query K' = {`Location`, `Dakota`, `Actor`, `Washington`}.

4. TRANSLATION OF KEYWORD QUERIES TO SPARQL QUERIES

4.1 Overview of the Translation Algorithm

The translation algorithm accepts a keyword-based query K and an RDF dataset T, and outputs a SPARQL query Q, which is a correct interpretation for K, in the sense that any result of Q is an answer for K over T. It assumes that T follows a simple RDF schema S.

In what follows, let G_T denote the RDF graph corresponding to T and D_S denote the RDF schema diagram of S.

Given a set of metadata matches $\boldsymbol{MM}[K,T]$ and a set of property value matches $\boldsymbol{VM}[K,T]$, we define two functions that group all keywords that match the same class or property:

$\boldsymbol{mm}[K,T] : \boldsymbol{IRI} \rightarrow 2^K$ such that
$\boldsymbol{mm}[K,T](r) = \{k \in K \,/\, (\exists p \in \boldsymbol{IRI})(\exists v \in \boldsymbol{L})((k,(r,p,v)) \in \boldsymbol{MM}[K,T])\}$

$\boldsymbol{vm}[K,T] : \boldsymbol{IRI} \rightarrow 2^K$ such that
$\boldsymbol{vm}[K,T](q) = \{k \in K \,/\, (\exists s \in \boldsymbol{IRI})(\exists v \in \boldsymbol{L})((k,(s,q,v)) \in \boldsymbol{VM}[K,T])\}$

We then define a *nucleus* as a triple $N=(C,PL,PVL)$, where

(1) $C=(K_0,c)$, with $K_0 = \mathbf{mm}[K,T](c)$, such that c is a class of S
(2) $PL=\{(K_1,p_1),\ldots,(K_m,p_m)\}$, with $K_i = \mathbf{mm}[K,T](p_i)$, such that c is the domain of a property p_i of S, for $i\in[1,m]$
(3) $PVL=\{(K_{m+1},q_1),\ldots,(K_{m+n},q_n)\}$, with $K_j = \mathbf{vm}[K,T](q_j)$, such that c is the domain of a property q_j of S, for $j\in[1,n]$

Note that, since there might be several keywords that match the same element of N, we consider sets of keywords, rather than a single keyword. Furthermore, since a keyword may match more than one element of N, we do not require that K_i and K_j be disjoint, for $0 \le i \ne j \le m+n$. We say that N *covers* the set of keywords $K_N = K_0 \cup K_1 \cup\ldots\cup K_m \cup K_{m+1} \cup\ldots\cup K_{m+n}$.

Given a set of nucleuses $\mathbf{N}=\{N_1,\ldots,N_m\}$, we also say that $\mathbf{N}$ *covers* $K_{N1}\cup\ldots\cup K_{Nm}$. Furthermore, we denote by $\mathbf{N_C}$ the set of classes of the nucleuses in $\mathbf{N}$ (which are nodes of D_S).

The translation algorithm implements two heuristics, called the *scoring* and the *minimization* heuristics. Intuitively, the scoring heuristic tries to capture the user intensions expressed in the list of keywords of a keyword-based query. Briefly, the scoring heuristic: (1) considers how good a match is, say "city" matches "Cities" better than "Sin City"; (2) assigns a higher score to metadata matches, on the grounds that, if the user specifies a keyword, say "city", that matches both a class label, say, "Cities", and the property value of an instance, say the film title "Sin City", then the user is probably more interested in the class labelled "Cities" than the specific film "Sin City"; (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, as follows.

Given a nucleus $N=(C,PL,PVL)$, the *score* of N, denoted $score(N)$, is the summation of all matches that N expresses, weighted by the type of the matches:

$$score(N) = (\alpha s_C + \beta s_P + (1-\alpha-\beta)s_V)$$

with

$$s_C = meta\_sim((K_0, c))$$

$$s_P = \sum_{(K_i,p_i)\in PL} meta\_sim((K_i, p_i))$$

$$s_V = \sum_{(K_j,q_j)\in PVL} value\_sim((K_j, q_j))$$

where

- α and β, with $0 < \alpha+\beta \le 1$, are parameters that weight between s_C, s_P and s_V, and which are experimentally set
- s_C is the combined score of the metadata matches for class c
- s_P is the combined score of the metadata matches for the properties in PL
- s_V is the combined score of the property value matches in PVL
- $meta\_sim((K,c))$ is the sum of metadata match scores of class c
- $meta\_sim((K,p_i))$ is the sum of metadata match scores of property p_i in PV
- $value\_sim((K,q_j))$ is the sum of property value match scores of property q_j in PVL

Section 4.2 illustrates how to estimate *meta_sim* and *value_sim* and compute the score of a nucleus.

The minimization heuristic tries to generate minimal answers, in two stages. Ideally, we 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 and generates a set $\mathbf{N}$ of nucleuses such that:

(1) $\mathbf{N}$ covers a large subset of K.

(2) All nodes in $\mathbf{N_C}$ are in the same connected component of D_S.

where, we recall, $\mathbf{N_C}$ denotes the set of classes of the nucleuses in $\mathbf{N}$ (which are again nodes of the RDF schema diagram D_S).

If we synthesized a SPARQL query Q based only on the nucleuses in $\mathbf{N}$, then an answer of Q – which would induce an answer for the keyword-based query K – would have as many connected components as there are classes in $\mathbf{N_C}$. Since answers are measured in terms of the number of nodes and connected components, this situation would be unsatisfactory. The second stage of the minimization heuristic then forces an answer to have a single connected component by connecting the classes in $\mathbf{C_N}$, using a small number of edges of D_S. This is equivalent to generating a Steiner tree ST of D_S whose nodes are the classes in $\mathbf{N_C}$. Then, the algorithm uses the edges of ST to generate equijoin clauses of the SPARQL query Q in such a way that any answer of Q indeed has a single connected component.

Note that ST exists since all nodes in $\mathbf{N_C}$ belong to the same connected component of D_S, by (2). Furthermore, note that we use the RDF schema diagram D_S, and not the RDF dataset graph G_T. In fact, this is the only step of the algorithm that depends on the assumption that T has a schema S.

Figure 2 shows a high level description of the translation algorithm, while Section 4.2 illustrates the synthesis of a SPARQL query.

Step 1 removes stop words from K and matches the remaining elements in K with literals in T, creating a set of metadata matches $\mathbf{MM}[K,T]$ and a set of property value matches $\mathbf{VM}[K,T]$, as defined in Section 3.2. Step 1 uses auxiliary tables to speed up computing matches (see also Section 4.2). For each class declared in S, the ClassTable table stores the IRI, label, description and other property values declared in S for the class. The PropertyTable stores the property metadata, as for the classes. The JoinTable stores domains and ranges declared in S. A forth table, ValueTable, stores all distinct property value pairs that occur in T.

Step 2 uses $\mathbf{MM}[K,T]$ and $\mathbf{VM}[K,T]$ to compute a set $\mathbf{M}$ of nucleuses as follows. It first processes class metadata matches, generating *primary nucleuses*; all class metadata matches with the same class will be mapped to a single nucleus. Then, it processes property metadata matches, creating the property lists of the primary nucleuses, or generating *secondary nucleuses*, for properties whose domains are not in any primary nucleus; finally, it processes property value matches, creating the property value lists of the existing nucleuses, or generating new *secondary nucleuses*, again for those properties whose domains are not in any previously constructed nucleus.

Step 3 computes the score of each nucleus in $\mathbf{M}$, as defined above.

Step 4 corresponds to the first stage of the minimization heuristic and creates a set $\mathbf{N}$ of nucleuses as follows. It first adds to $\mathbf{N}$ the nucleus N_0 in $\mathbf{M}$ with the largest score, removing it from $\mathbf{M}$. Let H_0 be the connected component of the RDF schema diagram D_S that contains the class of N_0. It also removes from $\mathbf{M}$ all nucleuses whose classes are not in H_0. This guarantees that Step 5 will be able to run correctly. Let K_{N0} be the set of keywords covered by N_0. The keywords in K_{N0} need no longer be considered and are disregarded from the nucleuses remaining in $\mathbf{M}$, which therefore have their scores recomputed. Step 4 continues by adding to $\mathbf{N}$ the nucleus in $\mathbf{M}$ with the largest (recomputed) score that covers a keyword not covered by any of the nucleuses previously selected. Since such

Translation Algorithm:

Input: A keyword query K
An RDF dataset T, with a simple RDF schema S

Output: A SPARQL query Q over T

1. Keyword matching:
   1.1. Eliminate stop words from K.
   1.2. Match each keyword with the classes, properties and property values in G_T, returning the set of metadata matches $\boldsymbol{MM}[K,T]$ and the set of property value matches $\boldsymbol{VM}[K,T]$, as defined in Section 3.2.2.
2. Nucleus generation:
   2.1. $\boldsymbol{M}$ = empty.
   2.2. For each class c such that there is a class metadata match for c in $\boldsymbol{MM}[K,T]$, do:
      2.2.1. If a nucleus with class c does not exist in $\boldsymbol{M}$, /* a metadata match for d exists */
      add to $\boldsymbol{M}$ a primary nucleus $N = ((K_c,c),\emptyset,\emptyset)$, with $K_c = \boldsymbol{mm}[K,T](c)$. /* which implies $K_c \neq \emptyset$ */
   2.3. For each property p such that there is a property metadata match for p in $\boldsymbol{MM}[K,T]$, do:
      2.3.1. Let d be the domain of p. If a nucleus N with class d does not exist in $\boldsymbol{M}$, /* no primary nucleus exists for d */
      add to $\boldsymbol{M}$ a secondary nucleus $N = ((K_d,d),\emptyset,\emptyset)$, with $K_d = \emptyset$. /* which implies $K_d = \emptyset$ */
      2.3.2. Let N be the nucleus with class d. Add (K_p,p), with $K_p = \boldsymbol{mm}[K,T](p)$, to the property list of N.
   2.4. For each property q such that there is a property value match for q in $\boldsymbol{VM}[K,T]$, do:
      2.4.1. Let d be the domain of q. If a nucleus N with class d does not exist in $\boldsymbol{M}$, /* no nucleus exists for d */
      add to $\boldsymbol{M}$ a secondary nucleus $N= ((K_d,d),\emptyset,\emptyset)$, with $K_d = \emptyset$. /* which implies $K_d = \emptyset$ */
      2.4.2. Let N be the nucleus with class d. Add (K_q,q), with $K_q = \boldsymbol{vm}[K,T](q)$, to the property value list of N.
3. Nucleus score computation:
   3.1. Compute the score of each nucleus in $\boldsymbol{M}$.
4. Nucleus selection:
   4.1. Initialize a set $\boldsymbol{N}$ with the nucleus N_0 in $\boldsymbol{M}$ with the largest score and remove N_0 from $\boldsymbol{M}$.
   4.2. Let D_S be the RDF schema diagram of S and H_0 be the connected component of S that contains the class of N_0. Remove from $\boldsymbol{M}$ all nucleuses whose classes are not in H_0.
   4.3. Update the sets of keywords and scores of the remaining nucleuses in $\boldsymbol{M}$ by dropping the keywords covered by N_0.
   4.4. While there are keywords not covered by the nucleuses in $\boldsymbol{N}$ and
   there is a nucleus in $\boldsymbol{M}$ that covers such keywords do:
      4.4.1. Add to $\boldsymbol{N}$ the nucleus N_s in $\boldsymbol{M}$ with the largest score such that N_s covers such keywords.
      4.4.2. Remove N_s from $\boldsymbol{M}$.
      4.4.3. Update the sets of keywords and scores of the remaining nucleuses in $\boldsymbol{M}$ by dropping the keywords covered by N_s.
5. Steiner tree generation:
   5.1. Let D_S again be the RDF schema diagram of S.
   Compute a Steiner tree ST of D_S that contains the set of classes of the nucleuses in $\boldsymbol{N}$.
6. Synthesis of the SPARQL query Q:
   6.1. Construct the WHERE and the TARGET clauses of Q from the nodes and edges of ST and the nucleuses in $\boldsymbol{N}$.
   6.2. Return Q.

Figure 2. Outline of the Translation Algorithm.

nucleuses are selected from $\boldsymbol{M}$, they necessarily have a class that is in H_0. It stops when all keywords in K are covered by the nucleuses in $\boldsymbol{N}$, or when no nucleus in $\boldsymbol{M}$ covers an uncovered keyword.

Step 5 implements the second stage of the minimization heuristic. It computes an (approximated) minimal Steiner tree ST of the RDF schema diagram D_S that covers $\boldsymbol{N_C}$, the set of nodes that correspond to the classes of the nucleuses in $\boldsymbol{N}$.

Although not shown in Figure 2, Step 5 proceeds as follows. It first computes a new labelled directed graph G_N whose nodes are those in $\boldsymbol{N_C}$ and there is an edge (m,n) in G_N labelled with k iff the shortest path in the RDF schema diagram D_S connecting nodes m and n has length k. Then, Step 5 computes a minimal directed spanning tree TN for G_N. If no such directed spanning tree exists, then Step 5 tries to compute a minimal spanning tree TN for G_N, but ignoring the edge direction. TN will then induce the desired Steiner tree ST of D_S covering the nodes in $\boldsymbol{N_C}$ by simply replacing each edge of TN by the corresponding path in D_S.

Step 6 synthesizes a CONSTRUCT query Q such that:

(1) Q returns a subset of T.
(2) The WHERE clause of Q contains filters that correspond to the elements of the property value pairs of the nucleuses in $\boldsymbol{N}$.
(3) The WHERE clause of Q contains equijoin clauses that correspond to the edges in ST.

Section 4.2 illustrates the synthesis of SPARQL queries.

To conclude, we state a lemma that captures the correctness of the algorithm:

Lemma 2: Let T be an RDF dataset, S be the RDF schema of T and K be a keyword-based query. Let Q be the SPARQL query the translation algorithm outputs for K, T and S. Then, any result of Q is an answer for K over T with a single connected component.

Proof Sketch

Step 1 of translation algorithm computes all possible matches between keywords in K and the RDF dataset T. Step 2 constructs the nucleuses by combining the matches found in Step 1. Steps 3 and 4 create a set of nucleuses $\boldsymbol{N}$ such that $\boldsymbol{N}$ covers as many keywords as possible. Let $\boldsymbol{C_N}$ be the set of the classes of the

nucleuses in $\boldsymbol{N}$. Note that $\boldsymbol{C_N}$ can be viewed as a set of nodes of the RDF schema diagram D_S. Step 5 connects, as much as possible, the nodes in $\boldsymbol{C_N}$ by paths in D_S, generating a Steiner tree ST of D_S that covers all nodes in $\boldsymbol{C_N}$. Finally, let Q be the CONSTRUCT query synthesized in Step 6 and A be a result of Q. Then, A is a subset of T and, by the construction of $\boldsymbol{N}$ and the filters in the WHERE clause of Q, A matches as many keywords in K as possible. Hence, A is answer for K over T. Furthermore, since ST is a Steiner tree of D_S that covers all nodes in $\boldsymbol{C_N}$, by the construction of the equijoin clauses in the WHERE clause of Q, the result A of Q will have a single connected component. □

4.2 An Example the Translation Process

This section illustrates how the algorithm synthesizes a SPARQL query for the following keyword-based query K:

```
Well Submarine Sergipe Vertical Sample
```

Step 1 searches the auxiliary tables ClassTable, PropertyTable and ValueTable to find matches with the keywords in K. For example, the following SQL query processes `Sergipe` against the ValueTable auxiliary table, whose columns are Property, Domain and Value ("fuzzy" is an Oracle function):

```
1. SELECT DISTINCT Property
2. FROM ValueTable
3. WHERE CONTAINS (Value, 'fuzzy({sergipe}, 70, 1)', 1) > 0
```

For the industrial dataset, Step 1 returns the following matches:

- A class metadata match
  M_1 = (`Sample`, (Sample, rdfs:label, "Sample"))
- A class metadata match
  M_2 = (`Well`, (DomesticWell, rdfs:label, "Domestic Well"))
- A property value match
  M_3 = (`Vertical`, (s, DomesticWell#Direction, v))
  since `Vertical` matches some value v of property DomesticWell#Direction (with domain DomesticWell).
- Two property value matches
  M_4 = (`Sergipe`, (s', DomesticWell#Location, v'))
  M_5 = (`Submarine`, (s'', DomesticWell#Location, v''))
  since `Submarine` and `Sergipe` match some values v' and v'' of property DomesticWell#Location (with domain DomesticWell).

Step 2 then generates two nucleuses:

- A first nucleus with just class Sample, using match M_1:
  N_1 = ((`{Sample}`, Sample), ∅, ∅)
- A second nucleus with class DomesticWell, using match M_2, and a property value list using matches M_3, M_4 and M_5:
  N_2 = ((`{Well}`, DomesticWell), ∅,
  {(`{Vertical}`, DomesticWell#Direction),
  (`{Sergipe, Submarine}`, DomesticWell#Location)})

Step 3 computes the scores of nucleuses N_1 and N_2 as follows. The score of nucleus N_1 is simple the score of the match of the keyword `Sample` with the value of the class label, which is the string "Sample". The score of nucleus N_2 is given by:

$$score(N_2) = (\alpha s_C + \beta s_P + (1 - \alpha - \beta) s_V)$$

where

- s_C = *meta_sim*((`{Well}`, DomesticWell)), which is the score of the match of the keyword `Well` with the value of the class label, which is the string "Domestic Well"
- s_P = 0, since the property list of the nucleus is empty
- s_V = *value_sim*((`{Vertical}`, DomesticWell#Direction)) + *value_sim*((`{Submarine, Sergipe}`, DomesticWell#Location))

For example, the value of

value_sim((`{Submarine, Sergipe}`, DomesticWell#Location))

is estimated by the following SQL query over the ValueTable auxiliary table, whose columns again are Property, Domain and Value (the prefix "ex:" is fictitious to preserve confidentiality of the data and "fuzzy" and "accum" are Oracle functions):

```
1. SELECT
2.    SCORE(1)/LENGTH(REGEXP_REPLACE(Value,'[^a-zA-Z0-9 -]',''))
3.    as score
4. FROM ValueTable
5. WHERE
6.     Domain = 'ex:DomesticWell' AND
7.     Property = 'ex:DomesticWell#Location' AND
8.     CONTAINS (Value,
9.         'fuzzy({submarine}, 70, 1) accum fuzzy({sergipe}, 70, 1)', 1) > 0
10. ORDER BY score DESC
11. OFFSET 0 ROWS FETCH NEXT 1 ROWS ONLY
```

Step 4 then selects the two nucleuses and Step 5 constructs a simple Steiner tree with just two nodes, corresponding to classes Sample and DomesticWell, connected by one edge, labelled with the object property Sample#DomesticWellCode.

Step 6 generates the SPARQL query Q below (which again uses the fictitious prefix "ex:"):

```
1. SELECT  ?C0 ?C1 ?P0 ?P1
2.  (<http://xmlns.oracle.com/rdf/textScore>(1) AS ?score1)
3.  (<http://xmlns.oracle.com/rdf/textScore>(2) AS ?score2) .
4. WHERE
5.   { ?I_C1 <ex:Sample#DomesticWellCode> ?I_C0 .
6.     ?I_C0 <ex:DomesticWell#Direction> ?P0 .
7.     ?I_C0 <ex:/DomesticWell#Location> ?P1
8.     FILTER (http://xmlns.oracle.com/rdf/textContains(?P0,
9.                                                 "fuzzy({vertical}, 70, 1)", 1)
10.      || http://xmlns.oracle.com/rdf/textContains(?P1,
11.           "fuzzy({submarine}, 70, 1) accum fuzzy({sergipe}, 70, 1)", 2))
12.    ?I_C0 rdfs:label ?C0 .
13.    ?I_C1 rdfs:label ?C1
14.  }
15. ORDER BY DESC(?score1 + ?score2)
16. LIMIT   750
```

The TARGET clause in Line 1 returns a table with variable bindings (the SELECT form of the query results). Although we adopted the CONSTRUCT form of a query, which returns a set of triples, to explain the notion of a keyword-based query answer, users preferred to see the results as a table, as discussed in Section 4.3.

Step 6 constructs the WHERE clause of the SPARQL query as follows. The (only) edge of the Steiner tree, labelled with the object property Sample#DomesticWellCode, generates the triple pattern in Line 5. Note that, since the domain of Sample#DomesticWellCode is the class Sample and the range is the class DomesticWell, variables ?I_C1 and ?I_C0 will respectively bind to instances of these classes. Hence, it is not necessary to include triple patterns that force ?I_C1 to be of type Sample and ?I_C0 to be of type DomesticWell.

The property value list of nucleus N_2 generates the triple patterns in Lines 6 to 11. The triple pattern in Line 6 instantiates variable ?P0 with the value of property DomesticWell#Direction for instance ?I_C0. Likewise, the triple pattern in line 7 instantiates variable ?P1 with the value of property DomesticWell#Location for instance ?I_C0.

The FILTER declaration in lines 8 and 9 matches the keyword `Vertical` with the value in ?P0, using the Oracle fuzzy matching function with the appropriate parameters (70 and 1). The matching score is returned in the Oracle predefined variable ?score1 (which is indicated by the "1" that appears as the last parameter in line 9).

The FILTER declaration in Lines 10 and 11 matches one of the keywords `Submarine` or `Sergipe`, or both, with the value in ?P1, using the Oracle fuzzy matching function, with the appropriate parameters (70 and 1), and the accum parameter, to sum the matching scores, if indeed both keywords match the value in ?P1. The matching score is returned in the Oracle predefined variable ?score2 (which is indicated by the "2" that appears as the last parameter in line 11).

Lines 12 and 13 translate the URIs in ?l_C0 and ?l_C1 to labels, which are hopefully user-friendly, and bind them to ?C0 and ?C1.

Finally, lines 15 and 16 order the query results in descending order of the combined scores and limit the result to 750 lines.

4.3 User Interface

The user interface offers an auto-completion feature to help users formulate a keyword-based query, as in Figure 3a. The interface suggests new keywords based on the previous keywords, the RDF schema vocabulary, and the labels that are resource identifiers (such as the "Sergipe", the name of a state).

Since an answer *A* for a keyword-based query *K* over an RDF dataset *T* is formally a subset of *T*, it would be consistent to present *A* as a set of triples. However, this option proved to be inconvenient for the users, which are more familiar with tabular data, as in relational systems. We then implemented a user interface that presents the results of *K* by combining a table with the Steiner tree underlying the SPARQL query, as in Figure 3b. The user may also select additional properties to be included in the table, as in Fig. 3c.

Finally, the interface allows the user to specify a keyword-based query which includes *filters*, such as:

```
Sample with Top between 2000m and 3000m
```

A *simple filter* involves only comparison operators, expressed in symbolic form, such as "<", or using reserved words, such as "`between`", whereas a *complex filter* is a Boolean combination of simple filters, expressed using Boolean operators. A filter typically involves constants, perhaps with a unit of measure, such as "`2000m`"; the tool converts all constants to the unit of measure adopt for the property being filtered. The syntax of the filters is specified by a grammar defined in ANTLR4 (ANother Tool for Language Recognition) [16].

5. EXPERIMENTS

5.1 Experiment setup

All experiments were conducted using a RESTful Web application develop in Java. The app ran on a desktop machine with OS Windows 7 Ultimate, a quad-core processor Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, 4 GB of RAM. To store and manage the RDF data, we used the Oracle Spatial and Graph for Semantic Technologies of Oracle 12c [14], running on a quad-core machine with processor Intel(R) Core(TM) i5 CPU 660 @ 3.33GHz, 7GB of RAM, and 4096 KB of Cache size. The database was configured with a PGA size of 324 MB and an SGA size of 612 MB with 148 MB of cache size and 296 MB of buffer cache.

The label and description columns of the auxiliary tables (see Section 4.1) were indexed using the CREATE INDEX statement of Oracle Text [19] to facilitate full text search over the stored values.

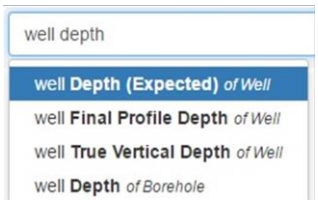


Figure 3a. Example of auto-completion.

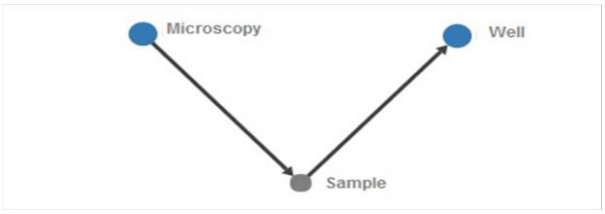


Figure 3b. Example of a query graph.

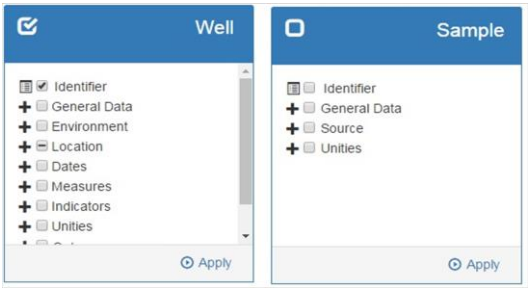


Figure 3c. Selection of additional properties.

In the case of RDF data, the Semantic Network feature of Oracle allows B-Tree indexing for RDF models and entailments [23].

5.2 Experiments with the Industrial Dataset

The data was originally stored in a conventional relational database, with well-documented tables and columns, which proved to be very helpful to identify metadata matches. The relational schema was normalized, as usual, which implies that a single table may represent several concepts and properties.

The triplification process used R2RML, the W3C standard RDB to RDF Mapping Language [6]. However, we soon realized that we had to capture additional metadata, such as which table columns were keys, which contained external names for the objects (such as state names and acronyms), etc. These additional metadata were important to guide keyword matches and to define how the object IDs were exposed to the users. Therefore, we proceeded as follows. First, on the relational side, we defined a set of views that denormalize the tables. Then, we created an XML document that defines all classes and properties of the RDF schema, as well as additional details, and that maps the RDF classes and properties one-to-one to the relational views. We developed a module that, using the XML document, generates the R2RML statements to map the relational data to triples and to load the auxiliary tables mentioned in Section 4.1.

Figure 4 shows a partial RDF schema diagram. The diagram depicts all classes (in rectangles), object properties (in single arrows, starting on the domain and ending on the range), with their names omitted to avoid cluttering the diagram, and subClassOf axioms (in

dashed arrows, starting on the sub-class and ending on the super-class).

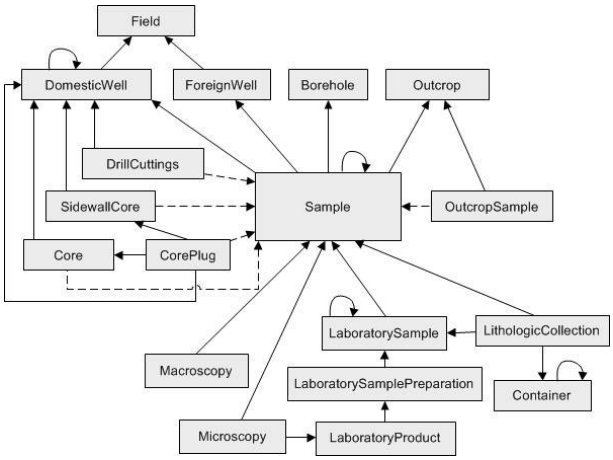


Figure 4. RDF schema of the industrial dataset.

The dataset is about hydrocarbon exploration (that is, oil and gas exploration). The instances of the central class, Sample, describe geological samples obtained during well drilling or directly from outcrops (rock formations visible on the surface). The instances of the classes DrillCuttings, SidewallCore, Core, CorePlug and OutcropSample correspond to sample sub-classes. The instances of the classes at the bottom of the diagram represent laboratory products, their macroscopic and microscopic analysis and where the products are stored.

The dataset has a large number of datatype properties (558). In particular, the values of the datatype properties of the instances of the classes Macroscopy and Microscopy are mostly literals, with a rich description of the laboratory products, which are highly amenable to keyword search. In fact, this motivated the project since users of the original relational database were mostly geologists, which were not happy with the relational database interface.

It took, on the average, 3 hours to triplify the relational database, generating an RDF dataset with about 130M triples (see Table 1), which implies that it is feasible to fully rematerialize the RDF dataset when needed, although we could have implemented an incremental rematerialization strategy.

We ran a suite of keyword-based queries to assess the performance of the tool, the correctness of the translation of the keyword-based queries and the adequacy of the result ranking. Table 2 (at the end of the paper) shows the runtime to process the keyword-based queries up until the first 75 answers were sent to the user, which corresponds to the first Web page (the time reported is the average of 10 executions for each sample query). The results show that all queries were successfully executed in less than 0.5 sec, which is quite reasonable, considering the size of the dataset.

Table 1. Statistics – Industrial dataset, IMDb and Mondial.

Triple Type	#Triples		
	Industrial	**IMDb**	**Mondial**
Class declarations	18	21	40
Object property declarations	26	24	62
Datatype property declarations	558	24	130
subClassOf axioms	5	-	-
Indexed properties	413	34	71
Distinct indexed prop instances	7.103.544	14.259.846	11.094
Class instances	8.981.679	72.973.275	43.869
Object property instances	11.072.953	184.818.637	63.652
Total triples	**130.058.210**	**395.394.424**	**235.387**

Finally, as a very preliminary user assessment, before early deployment, we asked 2 questions to 3 geologists to evaluate the same set of keyword-based queries. The results were very encouraging:

Question 1 (Correctness of the translation): "The results returned are a correct answer for the keyword-based query?"

Results: 8 x "Very Good", 9 x "Good" and 1 x "Regular.

Question 2 (Adequacy of the ranking of the results): "The expected results appear in the first Web page returned?"

Results: 6 x "Very Good", 11 x "Good" and 1 x "Regular.

Both "Regular" ratings were given by one of the users to the keyword-based query "`field exploration macroscopy microscopy lithologic collection`", which is fairly generic and returns a large number of answers.

We also opened the tool to a small user community, all of whom were quite surprised with the ease of use of the tool and the quality of the answers, and manifested their interest in expanding the tool to other Petrobras databases, which attests the success of the project.

5.3 Experiments with Mondial and IMDb

We tested the tool against triplified versions of the Mondial dataset (https://www.dbis.informatik.uni-goettingen.de/Mondial/) and IMDb (https://sites.google.com/site/ontopiswc13/home/imdb-mo). Contrasting with the versions adopted in Coffman's benchmark [4], these versions feature conceptual schemas with a complexity closer to the schema of the target industrial dataset (see Table 1). We used the same list of keyword queries as in Coffman's benchmark, albeit they are much simpler than those expected from the typical users of the industrial dataset. We ran all queries against each of these datasets and compared the results returned with the expected results (the full results are available at www.inf.puc-rio.br/~casanova/ under the Recent Tools section).

A summary of the results for the Mondial RDF dataset follows:

Queries 1-5 – countries: All queries correctly answered.

Queries 6-10 – cities: Queries correctly answered, except Query 6, which returned 2 results, since there are 2 cities named "Alexandria".

Query 11-15 – geographical: Queries correctly answered, except Query 12, which returned 2 results, since "Niger" is both a country and a river.

Queries 16-20 – organization: Some queries were not correctly answered since the expected values were not listed in class Organization (in the version of Mondial used).

Queries 21-25 – border between countries: Keywords match the labels of two instances of class Country; but the keywords are not sufficient to infer that the question is about the borders between countries and, thus, were not correctly answered.

Queries 26-35 – geopolitical or demographic information: Queries correctly answered, except Query 32.

Queries 36-45 – member organizations two countries belong to: The expected answer is the list of organizations that the countries belong to; however, the translation algorithm did not identify the IS_MEMBER class when generating the nucleuses.

Queries 46-50 – Miscellaneous: Some queries were successfully answered, while others were not, since the keywords do not always reflect the intended question.

A total of 32 queries, 64% of the 50 queries in Coffman's benchmark for Mondial, were correctly answered. As pointed out above, an analysis of the failed queries (see Table 3 for examples) reveals that: some may not be classified as failures (failed queries in the first 20 queries); some can be blamed to the lack of keyword semantics (failed queries in groups *21-25* and *46-50*); and some to the lack of accuracy of the keywords (as Query 50 in Table 3). These results actually indicate that the list of queries and query results in Coffman's benchmark should be reassessed.

Table 4 (also at the end of the paper) reports the results for the IMDb dataset. A total of 36 queries, 72% of the 50 queries in Coffman benchmark for IMDb, were correctly answered. Again, an analysis of the failed queries is instructive. For example, when running Query 41, we found a 1951 film with "Audrey Hepburn" in the title, rather than all 1951 films that the actress Audrey Hepburn starred. However, we would rather classify this result as a serendipitous discovery, rather than a failure.

6. CONCLUSIONS

We presented the results of an industrial project to facilitate access to a large relational database by combining RDF technology with keyword search. The algorithm to translate keyword-based queries to SPARQL queries takes advantage of the schema of the RDF dataset to avoid user intervention and achieve good performance, even for large RDF datasets. The user interface allows the user to specify keywords, as well as filters and unit measures. The interface presents the result of a keyword query with the help of tables, which is a familiar form of expressing query results, rather than as an RDF graph. Finally, the experiments covered both a real-world industrial dataset, as well as two familiar benchmarks. The tool proved to be quite robust to keyword-based queries over datasets with complex conceptual schemas, and not just toy schemas, which encourages its wider adoption at Petrobras, the industrial partner that supported the project reported in this paper.

As for future work, we plan to incorporate a domain ontology, being developed as a separated project, to expand keywords and therefore improve the usefulness of the tool. We also plan to allow filters with spatial operators. Lastly, we are working on a version of the application for a dataset federation.

7. ACKNOWLEDGMENTS

This work was partly funded by CNPq under grant 444976/2014-0, 303332/2013-1, 442338/2014-7 and 248743/2013-9 and by FAPERJ under grant E-26/201.337/2014. We would also like to thank the Petrobras team who help the authors design the tool.

8. REFERENCES

[1] Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, P., and Sudarshan, S. 2002. BANKS: Browsing and keyword searching in relational databases. VLDB 2002, 1083-1086.

[2] Agrawal, S., Chaudhuri, S. and Das, G. 2002. DBXplorer: A system for keyword-based search over relational databases. ICDE 2002, 5-16.

[3] Brickley, D. and Guha, R.V. (eds). 2014. RDF Schema 1.1. W3C Recommendation 25 February 2014.

[4] Coffman, J. and Weaver, A. 1999. An empirical performance evaluation of relational keyword search techniques. TKDE 1999.

[5] Cyganiak, R., David Wood, D. and Lanthaler, M. (eds.). 2014. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation (25 February 2014).

[6] Das, S., Sundara, S. and Cyganiak, R. (eds). 2012. R2RML: RDB to RDF Mapping Language. W3C Recommendation 27 September 2012.

[7] De Virgilio, R. 2012. RDF Keyword Search Query Processing via Tensor calculus. WWW 2012.

[8] De Virgilio, R. and Maccioni. A. 2014. Distributed Keyword Search over RDF via MapReduce. ESWC 2014, 208-223.

[9] Guo, Y., Pan, Z. and Heflin, J. 2004. LUBM: A Benchmark for OWL Knowledge Base Systems. J. Web Semantics 3(2-3), 158-182.

[10] Harris, S. and Seaborne, A. 2013. SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013.

[11] He, H., Wang, H., Yang, J. and Yu, P. 2007. Blinks: Ranked keyword searches on graphs. ACM SIGMOD 2007, 305-316.

[12] Hristidis, V. and Papakonstantinou, Y. 2002. DISCOVER: keyword search in relational databases. VLDB 2002, 670-681.

[13] Lanti, D., Rezk, M., Xiao, G. and Calvanese, D. 2015.The NPD Benchmark: Reality Check for OBDA Systems. EDBT 2015 – Industry and Applications.

[14] Murray, C. 2014. *Spatial and Graph RDF Semantic Graph Developer's Guide 12c*. Oracle. p. 636. E51611-06.

[15] Oliveira, P., Silva, A. and Moura, E. 2015. Ranking Candidate Networks of relations to improve keyword search over relational databases. ICDE 2015, 399-410

[16] Parr, T. 2013. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.

[17] Schmachtenberg, M., Bizer, C. and Paulheim, H. 2014. State of the LOD Cloud 2014 (Version 0.4, 08/30/2014) (available at: http://lod-cloud.net).

[18] Qiao, S. and Özsoyoğlu, Z. 2015. RBench: Application-Specific RDF Benchmarking. SIGMOD 2015, 1825-1838.

[19] Shea, C. 2014. *Oracle Text Reference, 12c Release 1 (12.1)*. Oracle. p. 718. E41399-05.

[20] Skjæveland, M., Giese, M., Hovland, D., Lian, E. and Waaler, A. 2015. Engineering ontology-based access to real-world data sources. J. Web Semantics: 33, 112-140.

[21] Tran, T., Wang, H., Rudolph, S. and Cimiano, P. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. ICDE 2009, 405-416.

[22] Vidal, V., Casanova, M., Neto, L.E. and Monteiro, J.M. 2014. A Semi-Automatic Approach for Generating Customized R2RML Mappings. ACM SAC 2014, 316-322.

[23] Wu, S., Perry, M. and Kolovski, V. 2010. *Oracle Database Semantic Technologies: Understanding How to Install, Load, Query and Inference*. Oracle.

[24] Yang, M., Ding, B., Chaudhuri, S. and Chakrabarti, K. 2014. Finding patterns in a knowledge base using keywords to compose table answers. VLDB Endow. 7(14), 1809-1820.

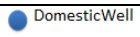
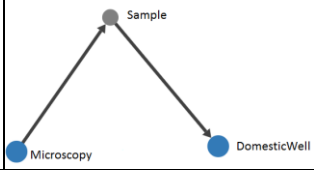
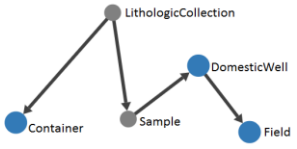
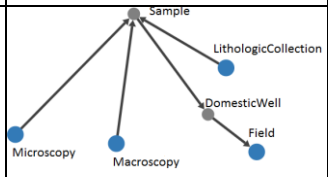
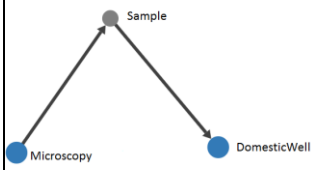
[25] Zenz, G., Zhou, X., Minack, E., Siberski, W. and Nejdl, W. 2009. From keywords to semantic queries - incremental query construction on the semantic web. J. Web Semantics 7(3), 166-176.

[26] Zhang, L., Tran, T. and Rettinger, A. 2013. Probabilistic Query Rewriting for Efficient and Effective Keyword Search on Graph Data. VLDB 2013, 1642-1653.

[27] Zheng, W., Zou, L., Peng, W., Yan, X., Song, S. and Zhao, D. 2016. Semantic SPARQL similarity search over RDF knowledge graphs. VLDB Endow. 9(11), 840-851.

[28] Zhou, Q., Wang, C., Xiong, M., Wang, H. and Yu, Y. 2007. SPARK: Adapting keyword query to semantic search. ISWC, 2007.

Table 2. Runtime to process sample keyword-based queries.

Keywords	Structure of the SPARQL query	Description of the nucleuses	Elapsed time (in milliseconds)		
			Query Synthesis	Query Execution	Total*
`well sergipe`		• A single nucleus with class DomesticWell • `sergipe` matches values of properties Basin, Localization, Federation, among others, of DomesticWell	15,4	446,3	**462,0**
`well salema`		• Two nucleuses with classes DomesticWell and Field, where the first one matches `well` • `salema` matches values of property Name of Field	25,0	246,4	**271,6**
`microscopy well sergipe`		• Two nucleuses with classes DomesticWell and Microscopy, which match the first two keywords • `sergipe` matches values of properties Localization, Basin, Federation, among others, of DomesticWell • The path from Microscopy to DomesticWell goes through the class Sample	23,2	327,3	**350,8**
`container well field salema`		• The first three keywords respectively match classes Container, DomesticWell and Field • `salema` matches values of property Name of Field • The non-directed path to join Container with DomesticWell and Field goes through Sample and LithologicCollection	24,3	315	**339,5**
`field exploration macroscopy microscopy lithologic collection`		• `exploration` matches values of properties OperativeUnit and AdministrativeUnit of class Field • According to the order they appear, the other keywords match classes Field, Macroscopy, Microscopy, and LithologicColletion • The paths leaving from Macroscopy, Microscopy, and LithologicColletion to Field go through the classes Sample and DomesticWell	43,8	180,1	**224,1**
`well coast distance < 1km microscopy bio-accumulated cadastral date between October 16, 2013 and October 18, 2013`		• Two nucleuses with classes DomesticWell and Microscopy • `coast distance` is a property of class DomesticWell filtered by the condition "`< 1km`" • `bio-accumulated` matches property Name of Microscopy • `cadastral date` is a property of class Microscopy, whose data type is date and is subjected to a filter • The path from Microscopy to DomesticWell goes through the class Sample	95,4	108,4	**204,1**

(*) Up to sending the first 75 answers.

Table 3. Selected queries from the Mondial Benchmark.

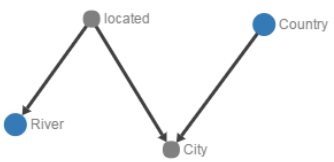
#Query	Keywords	Expected Answer	Application Answer	Observation
Query 16	`arab cooperation council`	Arab Cooperation Council	75 instances of class Organization	"Arab Cooperation Council" is not listed in class Organization (in the version of Mondial used)
Query 32	`Uzbekistan eastern orthodox`	Uzbekistan	-	"eastern orthodox" does not exist for property Name of class Religion (in the version of Mondial used)
Query 50: *Which Egyptian provinces does the Nile River flow through?*	`egypt nile`	Asyut Beni Suef El Giza El Minya El Qahira (munic.)	Egypt | Nile 	If the keyword `city` were added, we would correctly obtain: Egypt | Al Minya | Nile Egypt | Al Qahirah | Nile Egypt | Al Jizah | Nile Egypt | Bani Suwayf | Nile Egypt | Asyut | Nile

Table 4. Analysis of the IMDb Benchmark.

Queries 1-10: consist of the name of movie stars, such as "denzel washington". Relevant results contain a single tuple from the person relation that is the tuple of the specified individual.
Accuracy: 10 of 10. The result contained more than one tuple, if the movie star's name matched one of the keywords, but the top result was the expected actor.

Queries 11-20: consist of the name of movies, such as "gone with the wind". Relevant results contain a single tuple from the title relation that is the tuple of the specified film.
Accuracy: **9 of 10**. Again, the result contained more than one tuple, if the movie name matched one of the keywords, but the top result was the expected movie.
Error in Query 13 – "casablanca". "casablanca" is the name of a movie and of an actor; the score for both values was the same, but the algorithm returned the name of the actor, since the Actor class had a higher score than the Movies class. The movie name was the second generated query.

Queries 21-30: consist of the keyword "'title'" plus the name of film characters, such as "title atticus finch". Relevant results contain 3 tuples (1 from the char_name relation, 1 from the cast_info relation, and 1 from the title relation) that link the character to the film(s) in which s/he appears. (The keyword "title" is intentionally added to differentiate this group of topics from topics 1-20)
Accuracy: **7 of 10.** Again, the result contained more than one tuple.
Error in Queries 22, 23. The name of the character is part of the name of some title. The nucleus with class Title contained all keywords and had the best score. The answers of the algorithm were the titles with the character names.
Error in Query 28. In this case, the class AKA_TITLE has "darth vader" in one of its values. This nucleus was the best scored because the label of the class had the keyword "title" and "darth vader" as a value. Class Title only matched the keyword "title" and class char_name only matched "darth vader".

Queries 31-35: consist of the keyword "'title'" plus a film quote, such as "title frankly my dear i don't give a damn". Relevant results contain 2 tuples (1 from the movie_info relation and 1 from the title relation) that link the movie quote to the film in which it appears. (The keyword "title" is intentionally added so that relevant results answer the question "In which film does this quote appear?".) Note that a quote may appear in multiple films.
Accuracy: 4 of 5. The result was not a single tuple, as in previous blocks.
Error in Query 32: The quotes were not in the dataset used for the tests.

Query 36 "mark hamill luke skywalker". Relevant results must denote the films in which the actor Mark Hamill plays the character Luke Skywalker.
Accuracy: 1 of 1

Query 37 "tom hanks 2004": Relevant results contain 3 tuples (name <- cast_info -> title) that must denote all films in which the actor Tom Hanks appeared in the year 2004.
Accuracy: 1 of 1

Queries 38-40: Relevant results must denote the character that an actor plays in a film, such as "henry fonda yours mine ours char_name"
Accuracy: 1 of 3
Error in Queries 38 and 39: There are values in char_name that match "Henry Fonda" and "Russell Crowe". The algorithm assumed that the query was about these character names and tested with the movie name.

Query 41 "audrey hepburn 1951": Relevant results contain 3 tuples (name <- cast_info -> title) that must denote all films in which the actor Audry Hepburn appeared in the year 1951.
Accuracy: 0 of 1
Error: The nucleus with Title covered all three keywords since there is a film whose name matches "Audrey Hepburn" and whose production year matches 1951.

Query 42 "name jacques clouseau": A relevant result must identify an actor who plays Jacques Clouseau in a movie.
Accuracy: 0 of 2
Error: The algorithm found only the nucleus with class char_name, the character name matched with property name, and the keyword "name" matched with the label of the nucleus.

Query 44 "rocky stallone": Relevant results must denote a film in which Sylvester Stallone plays the character Rocky. Note that because of limitations of existing systems, relevant results are *not* required to include the appropriate tuple from the title relation (which would prevent any system from identifying a single relevant result).
Accuracy: 0 of 1
Error: the keywords are very ambiguous. The algorithm found both keywords in a PERSON_INFO#INFO value.

Query 45 "name terminator": A relevant result must identify an actor who plays "The Terminator"
Accuracy: 0 of 1
Error: same as for Queries 42-43.

Queries 46-49: Relevant results identify relationships (through the title relation) between an actor and another class, such as "harrison ford george lucas".
Accuracy: 3 of 4
Error in Query 48: "wachowski" only had matches in the AKA_NAME class.

Query 50 "indiana jones last crusade lost ark": Relevant results identify cast members in common between the films "Raiders of the Lost Ark" and "Indiana Jones and the Last Crusade."
Accuracy: 0 of 1
Error: The algorithm did not return the actors that both movies had in common, but returned the movies themselves.