Using Patterns for Keyword Search in RDF Graphs*

Hanane Ouksili
DAVID lab., Univ. Versailles
hanane.ouksili@uvsq.fr

Zoubida Kedad
DAVID lab., Univ. Versailles
zoubida.kedad@uvsq.fr

Stéphane Lopes
DAVID lab., Univ. Versailles
stephane.lopes@uvsq.fr

Sylvaine Nugier
EDF R&D, Chatou, France
sylvaine.nugier@edf.fr

ABSTRACT

An increasing number of RDF datasets are available on the Web. Querying RDF data requires the knowledge of a query language such as SPARQL; it also requires some information describing the content of these datasets. The goal of our work is to facilitate the querying of RDF datasets, and we present an approach for enabling users to search in RDF data using keywords. We introduce the notion of pattern to integrate external knowledge in the search process, which increases the quality of the results.

Keywords

RDF data, Keyword search, Patterns, Domain knowledge.

1. INTRODUCTION

A huge volume of RDF datasets is available on the Web, enabling the design of novel intelligent applications. In order to query these datasets, users must have information about their schema to target the relevant resources and properties. They must also be familiar with a formal query language such as SPARQL. An alternative approach to extract information from RDF datasets is keyword search, which is a straightforward and intuitive way of querying these datasets.

To illustrate our motivation, let us consider the RDF graph illustrated in Figure 1, containing data from AIFB institute[1], Karlsruhe University.

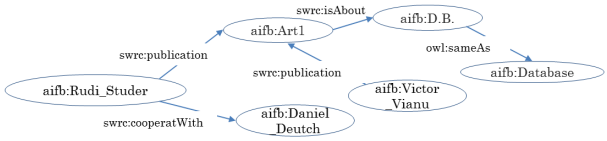


Figure 1: RDF subgraph from AIFB data

*This work was supported by Electricity of France (EDF), STEP department and the french ANR project CAIR.

[1]http://km.aifb.uni-karlsruhe.de/ws/eon2006/ontoeval.zip

Let "*Studer Cooperator*" be the keyword query which a user issues to find the scientists who have collaborated with a scientist named "*Studer*". To answer this query, existing approaches typically use a keyword-to-graph mapping function to identify a set of elements that contain the query keywords. Then, a connected "minimal" subgraph, which contains the identified elements is returned as an answer. Using such mapping function on the example, two elements will be identified: the resource $aifb$:$Rudi\_Studer$ and the property $swrc$:$cooperateWith$. The only result will be the scientist $aifb$:$Daniel\_Deutch$, which is linked to $aifb$:$Rudi\_Studer$ by the property $swrc$:$cooperateWith$. However, $aifb$:$Victor\_Vianu$ is also his collaborator. Indeed, they authored the same paper as shown in the graph. Some relevant answers will never be returned using the above process because the existing approaches use the structure of the graph and the linguistic content only. Moreover, the data is often incomplete and all property values are not necessarily present for all resources.

In this paper, we introduce a novel keyword search approach which returns graphs as a result to a keyword query. The key contribution of this paper is the use of external knowledge, expressed as patterns, during the extraction of the relevant graph fragments, the construction of the result and the ranking step. Experiments show that our approach delivers high-quality search results.

The rest of this paper is organized as follows. The concept of pattern is defined in section 2. We detail our solution for keyword search integrating patterns in Section 3. Section 4 reports the experimental results. Section 5 reviews the related works. Finally, we conclude the paper and present some future works in Section 6.

2. EXPRESSING KNOWLEDGE USING PATTERNS

The keywords used to query a dataset can not always be mapped into a graph element because the corresponding information might be missing from the dataset. If we take into account some external knowledge available for the considered domain, equivalence relations can be found between the terms used in the query and the elements of a dataset.

For example, let us consider the AIFB dataset, the $swrc$:$cooperateWith$ property between two researchers r_i and r_j states that they have already collaborated. If this property is missing for a pair of researchers but if we know that they wrote the same paper, we can infer that they have collaborated, even if the $swrc$:$cooperateWith$ property is missing. In order to capture these equivalence relations, we introduce

the notion of pattern, defined below.

2.1 Pattern Definition

The definition of patterns relies on the definition of *property expression* and *path expression* which are presented hereafter.

A *property expression* denoted *exp* is a triple (X, p, Y), where X and Y are either resources, literals or variables and p is a property. For example, $(X, swrc{:}isAbout, Database)$ is a *property expression* that represents triples having *Database* as object and *swrc:isAbout* as predicate.

A *path expression* describes a relation (possibly indirect) between resources. The relation is a path (a sequence of properties) between two resources. Note that a property expression is a special case of path expression where the path length is 1. More formally, a *path expression* denoted exP is defined as a triple (X, P, Y), where X and Y are either resources, literals or variables and P is a SPARQL 1.1 path expression[2]. For example,

$$(aifb{:}D.B., (owl{:}sameAs|\hat{}owl{:}sameAs)^{+}, Y)$$

represents all the resources related to $aifb{:}D.B.$ by a sequence of properties which are either $owl{:}sameAs$ or its inverse.

A *pattern* is a pair $[exp, exP]$ where *exp* is a property expression and exP is a path expression. It represents the equivalence between two expressions, one property expression and one path expression.

As an example, consider the RDF graph of Figure 1; the two resources $aifb{:}D.B.$ and $aifb{:}Database$ are related by the property $owl{:}sameAs$ which indicates that they are equivalent. The value of the property $swrc{:}isAbout$ for the resource $aifb{:}Art1$ is $aifb{:}D.B.$, therefore, this property has also the value $aifb{:}Database$. To express this knowledge, the following pattern is defined:

```
[(X, swrc:isAbout, Y), (X, swrc:isAbout/
                (owl:sameAs|^owl:sameAs)+, Y)]
```

This pattern represents the fact that the property *swrc:isAbout* is equivalent to a path composed by a property *swrc:isAbout* followed by a sequence of *owl:sameAs* properties.

Patterns can be used to capture external knowledge which can be user specific, domain specific or generic. Generic patterns are valid for any dataset as they are related to RDF, RDFS or OWL vocabularies.

2.2 Pattern Evaluation

Patterns are evaluated on the considered RDF graph to find a set of triples which satisfy the description given in the path expression. This set of triples will be stored and used during keyword search.

Let consider the subgraph represented in the Figure 1 and the following pattern which indicates that if two scientists have authored the same paper, then they have collaborated:

```
[(X, swrc:cooperateWith, Y),
(X, swrc:publication/^swrc:publication, Y)]
```

The evaluation of this pattern on the subgraph of Figure 1 results in the following triple:

[2]http://www.w3.org/TR/sparql11-property-paths/

```
{<aifb:Rudi_Studer, swrc:cooperateWith,
                        aifb:Victor_Vianu>}
```

We have introduced other properties to express other kinds of knowledge. For example, to express that a resource r_i contains some information which complements the one provided by another resource r_j, we have introduced a new property called *pattern:sameResult*. Consider the case of two resources related by a sequence of *owl:sameAs* properties. We can state that they contain complementary information as they represent in fact the same object. This is captured by the following pattern:

```
[(X, pattern:sameResult, Y),
        (X, (owl:sameAs|^owl:sameAs)+, Y)]
```

The result of its evaluation on the graph of Figure 1 is:

```
{<aifb:B.D., pattern:sameResult,
                        aifb:Database>}
```

3. KEYWORD SEARCH USING PATTERNS

In this section, we provide an overview of our approach, presented in Figure 2.

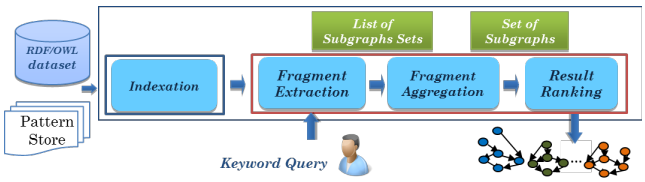


Figure 2: Keyword search process

The general framework contains four main components: (a) *Indexation*, which is performed independently from the query, pre-computes some information to speedup query processing; (b) *Fragment Extraction* uses the index and a mapping function to identify a set of elements (either nodes or edges) that contain the query keywords; an expansion is also applied in order to identify the relevant fragments of the graph according to the external knowledge formalized as patterns which are stored in the *pattern store*; (c) *Fragments Aggregation* merges the fragments to form a subgraph; finally (d) *Results Ranking* returns a list of ranked subgraphs representing the results. In the remaining of this section, each component will be detailed, and the underlying algorithms will be presented.

3.1 Index Construction

An inverted index is used to improve the efficiency of the relevant fragment extraction phase. Our index contains a set of documents, each one representing a graph element (resource, literal or property). The document structure is illustrated in Table 1.

The *Element* field contains the element of the graph which is represented, that is, a URI for resources and properties or a literal. *Type* indicates the type of the element: it can be a class, an instance, a literal, or a property. *Content* contains the text that was extracted from the graph element. It represents different parts of the element according to its type. To obtain the textual content of each document, we consider a representative keyword from the local name of URIs for resources and properties as document content; for example,

Element	Type	content	Fragment
"MULO Project"	literal	project mulo	<*aifb:proj1, swrc:name, "MULO Project"*>
swrc:Project	class	project	*swrc:Project*
swrc:worksAtProject	property	work project	$< aifb{:}Hartmut\_Schmek,\ swrc{:}worksAtProject,\ aifb{:}proj1 >$
swrc:worksAtProject	property	work project	$< aifb{:}Jurgen\_Brauke,\ swrc{:}worksAtProject,\ aifb{:}proj1 >$
swrc:worksAtProject	property	work project	$< aifb{:}H\_S,\ swrc{:}worksAtProject,\ aifb{:}proj2 >$

Table 1: Graphical representation of the document structure

a document is created for the property *swrc:hasAuthor*, containing the word *Author* (which can be generated using Information Extraction techniques). For literals, we consider their content as a document. Finally, *Fragment* represents the part of the graph which is relevant to the keyword, it is a subgraph that will be used to construct the result to be returned to the user. Instanciating this last field differs from one type of element to another. For all resources, we consider the corresponding node and we save its URI. In the case of a literal, the triple $< r_i,\ p_i,\ l >$ is saved, where l is the considered literal and r_i is the resource that it describes. For properties, one document is created for each triple $< r_i,\ pr,\ r_j >$, where pr is the considered property. This triple will be the fragment corresponding to the document. In Table 1, the first document represents the literal "$MULOProject$", the second represents the class $swrc{:}Project$ and others represent the property $swrc{:}worksAtProject$.

3.2 Extracting Relevant Fragments

Extracting relevant fragments consists in retrieving the parts of the RDF graph that correspond to the keywords of the query. Fragment extraction is composed of two steps.

Graph-query matching retrieves the graph elements containing one of the query keywords in their textual content by using the index. To this end, a mapping function and some standard transformation functions such as abbreviation and synonym are used. More formally, let G be the RDF graph and $Q = \{k_1, ..., k_n\}$ the keyword query. The goal is to return a set of lists $El = \{El_1, ..., EL_n\}$ where El_i is the list of fragments el_{ij} of the graph G which corresponds to the query keyword k_i. We refer to these as *keyword elements*. In addition, note that if the keyword element is a property as in our example, our approach will not extract all occurrences of the property in the graph but only the ones for which either the object or the subject is a relevant fragment. Otherwise, the result might include irrelevant answers. Indeed, if the query is "*cooperator Schemek*", we will select the property $swrc{:}cooperateWith$ of the resource representing the scientist *Schemek* to get only his cooperators and not cooperators of other scientists.

The expansion of the results allows to integrate parts of the graph that are relevant to the query even if they do not contain the query keywords. The expansion is done using external knowledge formalized as patterns and the result of their evaluation on the data graph. We use mainly three types of patterns for the expansion of the results. In the following, we detail each of these types and show how they can be instantiated and how they are used when extracting the fragments.

3.2.1 Patterns "sameResult"

In some cases, the relevant fragments selected using the graph-query matching do not provide the complete information that responds to the keywords issued by the user. For example, if the user's keyword is "*publication*", he expects to get instances of the class $swrc{:}Publication$. However, the matching identifies only this class as a relevant fragment.

We define a new property, $pattern{:}sameResult$, that we will use to create patterns describing the semantic closeness between two nodes of the graph. This pattern has the following form:

$$[(\mathrm{X},\ \mathrm{pattern:sameResult},\ \mathrm{Y}),\ (\mathrm{X},\ \mathrm{P},\ \mathrm{Y})]$$

The $pattern{:}sameResult$ property is used between two nodes x and y (resources or literals) to indicate that the information provided by x complements the information provided by y. If there is a triple

$$tr =< x,\ pattern{:}sameResult,\ y >$$

in the pattern store, and y is a relevant fragment for a keyword k, then the subgraph $(X,\ P,\ y)$ is returned as a relevant fragment for k instead of y alone, given that the evaluation of the pattern $[(X,\ pattern{:}sameResult,\ Y), (X,\ P,\ Y)]$ has generated the triple tr.

The expansion process using triples of the form $< x,\ pattern{:}\ sameResult,\ y >$ is described in Algorithm 1. Let α be the keyword fragment, β another resource of the graph, such that $tr_2 =< \beta,\ pattern{:}sameResult,\ \alpha >$ is a triple of the pattern store. The expansion step will construct a relvant fragment by replacing the resource α by the subgraph $(\beta,\ P,\ \alpha)$, consisting of the two resources α and β as well as the path described in the expression P which has led to the generation of tr_2.

Input: L: triples list of the pattern store,
$El = \{El_1, ..., EL_n\}$: relevant fragment list
Output: $El^e = \{El_1^e, ..., EL_n^e\}$: relevant fragment list after expansion
1: $EL^e \leftarrow EL$
2: **for all** $EL_i \in EL$ **do**
3: **for all** $el_{ij} \in EL_i$ **do**
4: **if** $\exists\ x \mid < x,\ pattern{:}sameResult,\ el_{ij} > \in L$ **then**
5: $EL_i^e \leftarrow EL_i^e \backslash el_{ij}$
6: $EL_i^e \leftarrow EL_i^e \cup (x,\ P,\ el_{ij})$ /*P is a pattern's path expression which generated the triple $< x,\ pattern{:}sameResult,\ el_{ij} >$ */
7: **end if**
8: **end for**
9: **end for**

Algorithm 1: Result expansion using $pattern{:}sameResult$

It is possible to instantiate this pattern with any property, according to the user's needs. In this paper, we describe the generic patterns using properties of the RDF, RDFS and OWL vocabularies, for example: $rdfs{:}subClassOf$, $rdf{:}type$ and $owl{:}sameAs$.

For the *owl*:*sameAs* property, the defined pattern leads to identify as a relevant fragment a subgraph composed of x, the keyword element y and the path relating them, that is composed of a sequence of *owl*:*sameAs* properties because x and y are equivalent. The patterns is as follows:

$$[(X,\ \mathrm{pattern:sameResult},\ Y),\ (X,\ (\mathrm{owl:sameAs}\,|\,\hat{}\,\mathrm{owl:sameAs})^{+},\ Y)]$$

For the *rdfs*:*subClassOf* and *rdf*:*type* properties, we define one pattern which considers x and the keyword element y and the path relating them as a relevant fragment because x is an instance or subclass of a keyword element y. The defined pattern is as follows:

$$[(X,\ \mathrm{pattern:sameResult},\ Y),\ (X,\ \mathrm{rdf:type?/rdfs:subClassOf}^{*},\ Y)]$$

These patterns are generic and can be used for all data sources. Other domain-specific patterns can be defined. For example, in the case of the SWRC data source, if the user wants to have the supervisor and his/her student and considering that the latter is a keyword element, then the following pattern can be defined:

$$[(X,\ \mathrm{pattern:sameResult},\ Y),\ (X,\ \mathrm{swrc:supervises},\ Y)]$$

Figure 3 illustrates two relevant fragments after expansion using patterns. The colored resources represent keyword elements. $Frag_1$ is obtained after expansion using the properties *rdfs*:*subClassOf* and *rdf*:*type*, and $Frag_2$ using the *owl*:*sameAs* property.

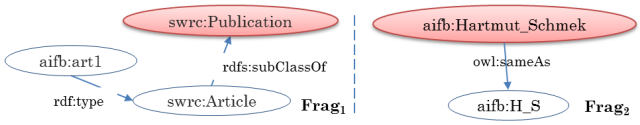


Figure 3: Two relevant fragments after expansion

3.2.2 *Patterns "sameProperty"*

The RDF, RDFS and OWL vocabularies are used in this category of patterns to identify properties that can replace other properties in the graph according to the schema of the domain being studied. Let *pr* be a resource representing a property, i.e. which has *rdf*:*Property* as the value for the *rdf*:*type* property. This resource can be linked to other resources of type *rdf*:*Property* with properties indicating a semantic closeness between them.

Figure 4 shows an excerpt of RDF graph with its schema. We can see that property types are interlinked. The closeness between property types must be reflected on the properties in the data. For example, we can take into account that the *swrc*:*member* property of the resource *aifb*:*Hartmut_Schmek* can be replaced by the *swrc*:*employee* property because this latter is the generalization of the former as indicated in the schema part. Therefore, if *swrc*:*member* property is relevant, then the *swrc*:*employee* is also relevant.

The new property *pattern*:*sameProperty* is used in the definition of patterns having the following form:

$$[(X,\ \mathrm{pattern:sameProperty},\ Y),\ (X,\ P,\ Y)]$$

The *pattern*:*sameProperty* property is used between two nodes x and y representing the property types to indicate

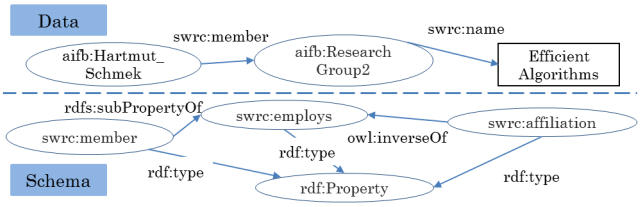


Figure 4: RDF subgraph with its schema

that a property of type y can be replaced by a property of type x. Therefore, if the triple $< x,\ pattern$:*sameProperty*$,\ y >$ is in the pattern store, and if y is a relevant property then x is also a relevant property.

Algorithm 2 describes this type of expansion: the set of relevant fragments is scanned and for every relevant element el_{ij}, the algorithm checks for a triple of the form $< Pr,\ pat$:*sameProperty*$,\ el_{ij} >$ in the pattern store (line 4). The existence of this triple means that the *pr* property is also relevant and is therefore added to the list of relevant items (line 5). In addition, the algorithm searches in the RDF graph for triples which contain properties of type *pr* to add them to the list of relevant fragments (line 8). As seen previously, properties are relevant if their subject or object are relevant fragments. The same constraint holds for the selection of triples during the expansion (line 7).

Input: L: triples list of the pattern store, G: RDF graph, $El = \{El_1, ..., EL_n\}$: relevant fragment list
Output: $El^e = \{El_1^e, ..., EL_n^e\}$: relevant fragment list after expansion

1: $EL^e \leftarrow EL$
2: **for all** $EL_i \in EL$ **do**
3: **for all** $el_{ij} \in EL_i$ **do**
4: **if** $\exists\ pr\ |\ < pr,\ pat$:*sameProperty*$,\ el_{ij} > \in L$ **then**
5: $EL_i^e \leftarrow EL_i^e \cup \{pr\}$
6: **for all** $< x,\ pr,\ y > \in G$ **do**
7: **if** $\exists\ EL_k\ |k \neq i\ \wedge\ (x\ \in EL_k \vee y \in EL_k)$ **then**
8: $EL_i^e \leftarrow EL_i^e \cup \{< x,\ pr,\ y >\}$
9: **end if**
10: **end for**
11: **end if**
12: **end for**
13: **end for**

Algorithm 2: Result expansion using *pattern*:*sameProperty*

The following is an example of this kind of patterns where three properties are used: *owl*:*inverseOf*, *owl*:*equivalentProperty* and *rdfs*:*subPropertyOf*. The pattern allows to consider a property x as relevant if it is equivalent, inverse or sub-property of the keyword element y.

$$[(X,\ \mathrm{pat:sameProperty},\ Y),\ (X,\ (\mathrm{owl:inverseOf}\,|\,\hat{}\,\mathrm{owl:inverseOf}\,|\,\mathrm{owl:equivalentProperty}\,|\,\hat{}\,\mathrm{owl:equivalentProperty}\,|\,\mathrm{rdfs:subPropertyOf})^{+},\ Y)]$$

3.2.3 *Patterns using domain properties*

A property can be considered as relevant because its URI contains a query keyword. However, RDF graphs are often

incomplete and properties are not defined for all resources. In this case, the values of these properties can be derived using external knowledge. We use patterns to retrieve relevant fragments which do not necessarily contain the keywords of the query but which meet the user's needs.

Consider the following pattern: $[(X,\ p,\ Y),\ (X,\ P,\ Y)]$ and assume that the local name of the p property contains one of the query keywords. Assume also that there are two resources r_i and r_j in the RDF graph such that there is no triple $< r_i,\ p,\ r_j >$ in the graph. If this triple exists in the pattern store as a result of the evaluation of the patterns because the two resources r_i and r_j are linked by a path corresponding to the path expression P, then the subgraph $(r_i,\ P,\ r_j)$ is a relevant fragment.

Input: L: triples list of the pattern store, G: RDF graph, $El = \{El_1, ..., EL_n\}$: relevant fragment list
Output: $El^e = \{El_1^e, ..., EL_n^e\}$: relevant fragment list after expansion

1: **for all** $EL_i \in EL$ **do**
2: **for all** $el_{ij} \in EL_i$ **do**
3: **if** el_{ij} is a property **then**
4: /*$< el\_ij,\ rdf:type,\ owl:Property > \in G$ */
5: **for all** $< x,\ el_{ij},\ y > \in L$ **do**
6: **if** $\exists\ EL_k\ |k \neq i\ \wedge\ (x\ \in EL_k \vee y \in EL_k)$ **then**
7: $EL_i^e \leftarrow EL_i^e \cup (x,\ P,\ y)$ /*P is a pattern's path expression which generated the triple $< x,\ el_{ij},\ y >$ */
8: **end if**
9: **end for**
10: Note el_{ij} as seen
11: **end if**
12: **if** el_{ij} is like $< subject\_el_{ij},\ property\_el_{ij},\ object\_el_{ij} >$ **then**
13: **if** $property\_el_{ij}$ is not seen **then**
14: **for all** $< x,\ property\_el_{ij},\ y > \in L$ **do**
15: **if** $\exists\ EL_k\ |k \neq i\ \wedge\ (x\ \in EL_k \vee y \in EL_k)$ **then**
16: $EL_i^e \leftarrow EL_i^e \cup (x,\ P,\ y)$ /*P is a pattern's path expression which generated the triple $< x,\ el_{ij},\ y >$ */
17: **end if**
18: **end for**
19: **end if**
20: Note $property\_el_{ij}$ as seen
21: **end if**
22: **end for**
23: **end for**

Algorithm 3: Result expansion using domain properties

For example, the property *swrc:cooperateWith* between two researchers in the ontology SWRC states that they have already collaborated. However, this property is not always defined. If we know that they have authored the same paper, we can infer that they have collaborated, even if the property *swrc:cooperateWith* is missing. We formalise this using the following pattern:

```
[(X, swrc:cooperateWith, Y),
(X, swrc:publication/^swrc:publication, Y)]
```

This pattern is used by the *fragment extraction* module which considers the subgraph corresponding to the path (*swrc*: *publication/^swrc:publication*) as a relevant fragment if the property *swrc:cooperateWith* is a keyword element. Consider that *"Studer Cooperator"* is a query issued to find the collaborators of the researcher named *Studer* in the graph of Figure 1. Let us consider the following two extracted graph elements: the node $aifb$:*Rudi_Studer* and the property *swrc*:*cooperateWith*. Without using patterns, the only result will be the subgraph G_1 of Figure 5 representing *Daniel Deutch*, a collaborator of *Rudi Studer*, both linked by the *swrc:cooperateWith* property. However, *Victor Vianu* is also a collaborator of *Rudi Studer* as they have authored the same paper ($aifb$:$Art1$). Using the pattern defined above, our approach will also return the subgraph G_2 of Figure 5 as a result.

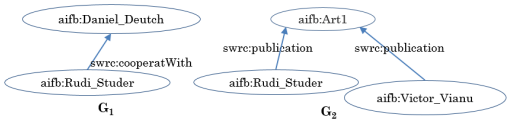


Figure 5: Keyword search results

Algorithm 3 describes this type of expansion: for every relevant fragment el_{ij}, if a triple of the form $< el_{ij}$, *rdf*:*type*, *rdf*:*Property* $>$ exists in the RDF graph (line 3), this means that el_{ij} is a resource representing a property. Thus, the algorithm finds triples of the form $< x,\ el_{ij},\ y >$ (line 5) from the pattern store. Indeed, in some cases, a relevant property exists in the schema as an instance of the *rdf*:*Property* class, but no resource is described using this property in the data graph. For example, in the graph of Figure 4, if the resource *swrc*:*employs*, which corresponds to a property, is relevant, then the algorithm will retrieve in the pattern store triples of the form $< x,\ swrc$:*employs*, $y >$. The algorithm will then search for each of these triples the corresponding subgraph in the graph G, in order to add it as a relevant fragment (line 7), provided that either x or y is a relevant fragment (line 6). In other cases, properties might be used in the data graph without being defined in the schema. As explained in Section 3.1, if a property is relevant, the relevant fragment is composed of all the triples containing this property. In this case, for each triple of the relevant fragment, if the corresponding predicate is not marked, then the algorithm searches for triples having this predicate in the pattern store (lines 12-14). For each of these triples, the corresponding subgraph in the graph G is retrieved, in order to add it as a relevant fragment (line 16), provided that either x or y is a relevant fragment (line 15).

3.3 Result Construction

After obtaining the set of relevant fragments for each query keyword, the *result construction* process builds the subgraphs representing the results. Each result is a connected minimal subgraph which contains for each keyword k_i one corresponding relevant fragment el_{ij}.

We perform the Cartesian product between the different sets of relevant fragments to construct the combinations. Then, for each combination, we look for minimal subgraphs to join different relevant fragments and construct the result. In other words, the result should contain the minimum intermediate nodes which are not relevant fragments. To do that, we use shortest path algorithms. Let consider two relevant fragments el_i and el_j. The shortest path between

the two fragments is the shortest path between two nodes $n_i \in el_i$ and $n_j \in el_j$ such that $\nexists\ n_{i2} \in el_i$, $\nexists\ n_{j2} \in el_j$, $n_i \neq n_{i2}$, $n_j \neq n_{j2}$ and $\mid (n_{i2} - -n_{j2}) \mid < \mid (n_i - -n_j) \mid$, where $\mid (x - -y) \mid$ is the size of the shortest path between nodes x and y. To construct the result, we combine the shortest paths between all pairs of fragments. We take into account the existing patterns to calculate the path between resources. If a pattern indicates that one path is equivalent to one property, the distance is reduced to 1.

3.4 Result Ranking

Since several graph elements may correspond to the same keyword, several subgraphs are returned to the user. We define a ranking function to evaluate the relevance degree of each subgraph. Our approach combines two criteria: (i) *Compactness Relevance* where compact answers are preferred; it increases when the size of the intermediate part decreases; and (ii) *Mapping Relevance* which is calculated using standard IR techniques. In our approach, we have used the TF function. The final score is the combination of the two scores and it is calculated as follows:

$$Score(R_i) = \alpha\ SizeScore(R_i) + (1 - \alpha)\ MappingScore(R_i)$$

Where $SizeScore(R_i)$ is the compactness relevance of the result R_i and $MappingScore(R_i)$ is the matching relevance. α is a parameter which expresses the weight of the two ranking criteria.

3.4.1 Compactness Relevance

The size of the intermediate part of the graph, added during the construction of the result, is an important element to take into account: the relevance of the result increases as the size of the intermediate part decreases. The size of a subgraph is the sum of the number of its edges and nodes. Moreover, in an RDF graph, the edges have a specific semantic. Consequently, the semantic closeness between resources does not only depend on the size of the path that connects them, but also on the semantic carried by the properties which form this path. For example, two resources linked by a path composed of a sequence of *owl:sameAs* properties are semantically closer than two other resources connected by a smaller path composed of other properties. Patterns are taken into account when calculating the size of a subgraph: if this subgraph is composed of a path defined in one of the patterns, then its size is reduced to the value 1. The real length of a path which represents a strong semantic link is not considered as it is, but is reduced. The compactness relevance is calculated as follows:

$$SizeScore(R_i) = \frac{size(\bigcup RelevantFragments)}{size(R_i)}$$

Where $size(\bigcup relevantfragments)$ is the size of the union of the relevant fragments and $size(R_i)$ is the size of the result subgraph R_i after the patterns are taken into account. If no intermediate part has been used, the size of the union of relevant fragments will be equal to the size of the result.

3.4.2 Matching Relevance

We use a classical information retrieval function to assign a weight to each relevant fragment of the result, namely TF (Term Frequency): the more frequent the keyword of the query in the document, the more relevant it is. The TF function also depends on the size of the document, the smaller the document containing the keyword of the query, the more it is focused on the term and therefore the more relevant it is. The equation below is used to obtain the weight of a relevant fragment:

$$TF(el_i, k_j) = \frac{frequence(k_j, el_i)}{number\_words(el_i)}$$

Where $frequency(k_j, el_i)$ is the frequency of the keyword k_j in the document representing the element el_i, and where $number\_words(el_i)$ is the number of words in the document representing the element El_i. The matching score of the result is obtained by calculating the average of the scores of the composing relevant fragments; it is computed as follows:

$$MappingScore(R_i) = \frac{1}{n} \sum_{j=1}^{n} TF(el_j, k_j)$$

Where k_j is a query keyword and n the number of keywords.

4. EXPERIMENTAL EVALUATIONS

We have implemented our approach in PatEx, a system for RDF data exploration [6]. We have developed our system as a desktop application implemented in Java. We have used the Jena API for the manipulation of RDF data and the Lucene[3] API to create the index and to search for keyword elements. The Jung[4] API is used for graph manipulation and visualization.

In this section, we describe our experiments to demonstrate the effectiveness of our approach. All experiments have been done on an Intel Core i5 with 12GB RAM. In the rest of this section, we first describe the used datasets and the algorithms to which we have compared our approach. Then, we discuss the results of our evaluations, related to the performance of our system and the quality of the results.

4.1 Experimental settings

Datasets.

We have used two datasets: AIFB and conference[5]. AIFB is dataset containing data taken from AIFB institute, Karlsruhe University. It is about entities of research communities such as persons, organizations, publications (bibliographic metadata) and their relationships. The dataset contains 8281 entities and 29 223 triples. The conference dataset contains information about the main conferences and workshops in the area of Semantic Web research, papers that were presented and people who attended these events. This dataset contains 571 entities and 1429 triples.

Algorithms.

We have compared our approach, which we refer to as WP, to two configurations which do not integrate patterns: (i) the first one, which we refer to as WAP-WAPr, considers node content only, (ii) the second one, which we refer to as WAP-WPr, considers both node and edge content, but without any condition during their extraction.

Metrics.

We have used two metrics to evaluate the quality of the results: Top-k precision ($P@k$) calculated with equation 1; it

[3]http://lucene.apache.org/core/
[4]http://jung.sourceforge.net/
[5]http://data.semanticweb.org/dumps/conferences/

provides the percentage of relevant results among the top-k results.

$$P@k = \frac{Number\ of\ relevant\ results}{k} \quad (1)$$

The second metric is the Normalized Discounted Cumulative Gain ($NDCG@k$); it is calculated using equation 2 [3]. This metric evaluates both the relevance of the results and their classification in the list. It is based on the idea that a relevant answer which is misclassified or at the end of the list is less useful to the user.

$$NDCG@k = \frac{1}{|Q|} \sum_{q \in Q} Z_k \sum_{k=1}^{k} \frac{2^r el(i) - 1}{log(i+1)} \quad (2)$$

Where Q is the set of used queries, $rel(i)$ is the relevance score of the result at position i and Z_k is a normalization parameter to get a score between 0 and 1.

We have also evaluated the search performance of our approach which is represented by the average time to find the top-k answers.

4.2 Efficiency Evaluation

We vary the query size from 2 to 5 keywords. For each size, the average execution time for 10 queries which include a wide variety of keywords for the three algorithm configurations are shown in Figures 6a and 6b.

We observe that the execution time increases almost linearly with the number of keywords for both datasets. The reason is that the number of combinations increases and therefore the number of results increases. But as the query size issued by users is usually small (2.35 in average according to [7]), the search performance of our approach is acceptable. However, we can see that the execution time is more important for the *AIFB* dataset than for the *Conference* dataset, because the size of the former is larger. In addition, the execution time of WP configuration is higher than WAP-WAPR, but less important than WAP-WPR. Indeed, WAP-WAPR does not consider properties and produces less results than the others. The WAP-WPR configuration considers properties and does not apply any filter to select only the relevant triples. The three configurations are in the same order of magnitude.

The execution time also depends on the type and number of keyword elements. Figure 6c shows the execution time for six queries, with different sizes, on the dataset *AIFB*. We can notice that the number of keywords in the query is not the only parameter that influences the execution time. Indeed, the query $Q_2(3 : 2L,\ 2L,\ 1L)$[6], which is longer than $Q_1(2 : 399L,\ 432L)$, takes less time. This is explained by the fact that Q_1 includes more general concepts and thus matches with more elements in the RDF graph. We conclude that the execution time depends on the number of keyword elements. We also notice that the execution time depends on the number of relevant fragments returned after the expansion. For example, $Q_6(2 : 1C\ and\ 4L,\ 1P)$ takes more time than $Q_4(2 : 1C,\ 1P)$. The pattern used in this case is to retrieve the instances of two classes $swrc{:}Person$ and $swrc{:}Fullprofessor$ and for the $swrc{:}cooperatewith$ property, it is the same processing for both queries. As the first class has more instances, we will have more results which is why it takes more time. For $Q_3(2 : 1L,\ 1P)$, we can observe that the WAP-WPR configuration takes more time than others because it matches with the $swrc{:}cooperatewith$ property and this configuration considers all triples having this property without any filter.

[6]The query contains three keywords, and matches elements: L for literals, C for classes ans P for properties

4.3 Effectiveness Evaluation

For these experiments, we have used 10 queries that are randomly constructed (with 2 to 5 keywords). We have compared the three configurations of the algorithm described in Section 4.1 and we have asked 4 users to assess the top-k results of each query using the three configurations on a 3-levels scale: 3 (perfectly relevant), 1 (relevant) and 0 (irrelevant). To calculate $P@k$, we assume that the scores 3 or 1 correspond to a relevant result and 0 to an irrelevant one.

We have used $NDCG@k$ and $P@k$ to evaluate the quality of the results (see Section 4.1). Table 2 reports the average $NDCG@k$ and $P@k$ for the 10 considered queries with k=5, 10 and 20 on the two datasets. We can see from this table that the use of patterns in the algorithm allows to significantly outperform the two other configurations in terms of $NDCG@k$ and $P@k$ values at all levels for both datasets and for all values of k. This means that our approach finds more relevant results. Furthermore, since $NDCG@k$ includes the ranking position of the results, our ranking algorithm is better because it includes patterns during the ranking step.

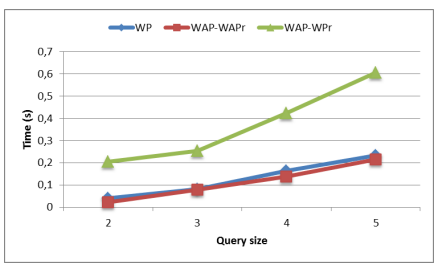
5. RELATED WORKS

Keyword search in RDF data has emerged as a new research topic [4, 10, 1, 5]. Among the most known systems are SPARK [14], NAGA [4] and Q2Semantic [10]. They return either SPARQL queries or subgraphs as answers for a given keyword query.

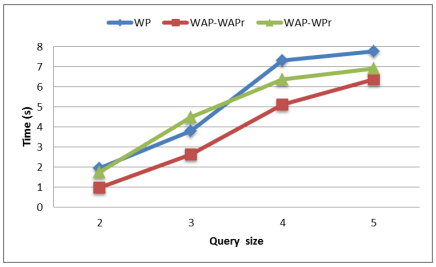
There are mainly two categories of approaches dealing with keyword search over RDF graphs. The first one uses information retrieval techniques. The proposed approaches first define what a document is, for example, a triple [1] or a resource with its neighborhood [2]. Documents are indexed, then query keywords are mapped to the defined documents. Finally, a ranked list of documents is returned to the user. These works do not merge the relevant fragments corresponding to the query keywords. Therefore, each result is often the answer to a part of the query.

In the second category, approaches use graphs to construct results [14, 11, 13]. To do so, they typically use a keyword-to-graph mapping function to identify a set of elements that contain the query keywords. Then, a connected "minimal" subgraph, which contains one identified element for each query keyword is returned as an answer. Since several elements can contain the same keyword, several results are returned to the user, and each approach defines a ranking function mainly based on the size of the subgraph, the matching relevance and the coverage. In these works, only the structure of the graph is considered during the construction of the result, unlike our approach, which integrates the semantic through patterns. In [8, 9], the authors have predefined templates for the results based on the ontology or the types of keyword elements. However, defining these templates might result in capturing some interpretations of the keywords which may not satisfy the user's needs.

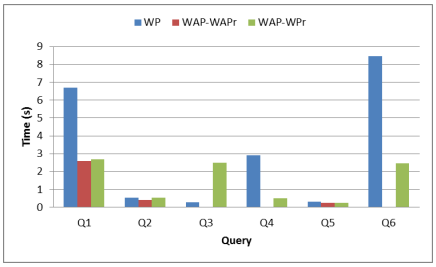
6. CONCLUSION AND FUTURE WORKS



(a) Effect of query size: Conference



(b) Effect of query size: AIFB



(c) Effect of keyword element type

Figure 6: Execution time evolution

Data	AIFB								
Algorithms	WP			WAPWPr			WAPWAPr		
k	5	10	20	5	10	20	5	10	20
P@k	0.99	0.98	0.97	0.73	0.73	0.69	0.66	0.66	0.66
NDCG@K	0.92	0.90	0.90	0.64	0.63	0.58	0.47	0.46	0.46
Data	Conference								
Algorithms	WP			WAPWPr			WAPWAPr		
k	5	10	20	5	10	20	5	10	20
P@k	0.98	0.98	0.96	0.91	0.87	0.81	0.86	0.86	0.82
NDCG@k	0.74	0.70	0.68	0.59	0.53	0.50	0.47	0.45	0.43

Table 2: Effectiveness evaluation results

This paper proposes a novel approach to query an RDF dataset using keywords; it differs from existing approaches in that it includes external knowledge. We have introduced the notion of pattern to formalise this knowledge. In our approach, an answer to a query is a set of sugraphs containing for each keyword, at least one relevant fragment. We have described how we can expand relevant fragments to complement them or to find other fragments using patterns. We have also developed a new index including all types of graph elements and we have used the patterns during the construction of the results as well as in the ranking function. The experiments show that the patterns improve the quality of the results. The concept of pattern used in this paper could be compared to the use of inference. However, the use of patterns allowed us to introduce knowledge based on the schema but also on the data without dealing with the types of entities or their hierarchy in the ontology. Moreover, we can instanciate one or both ends of the path expression by replacing the variables with resources or literals, which allows to focus on specific cases.

In future works, we will explore the summarisation of the results produced as an answer to a keyword query. As some results can have complementary information, we could aggregate them to obtain one result which is relevant to the query. Few works have addressed this problem in the case of keyword research in RDF graphs. The proposal presented in [12] is a contribution to solve this problem. We will study the introduction of patterns in the summarisation process to improve the results.

7. REFERENCES

[1] S. Elbassuoni and R. Blanco. Keyword search over rdf graphs. In *CIKM*, pages 237–242, 2011.

[2] R. Guha, R. McCool, and E. Miller. Semantic search. In *WWW*, pages 700–709, 2003.

[3] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, pages 41–48, 2000.

[4] G. Kasneci, F. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *ICDE*, pages 953–962, 2008.

[5] W. Le, F. Li, A. Kementsietsidis, and S. Duan. Scalable keyword search on large rdf data. *TKDE*, 26:2774–2788, 2014.

[6] H. Ouksili, Z. Kedad, S. Lopes, and S. Nugier. Patex: Pattern oriented RDF graphs exploration. In *NLDB*, pages 102–114, 2016.

[7] S. Oyama. *Query Refinement for Domain-Specific Web Search*. PhD thesis, kyoto University, 2002.

[8] C. Pradel, O. Haemmerlé, and N. Hernandez. Swip: A natural language to sparql interface implemented with sparql. In *ICCS*, pages 260–274, 2014.

[9] M. Rahoman and R. Ichise. Automatic inclusion of semantics over keyword-based linked data retrieval. *IEICE Transactions*, 97-D:2852–2862, 2014.

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

[11] H. F. Wang, K. Zhang, Q. L. Liu, T. Tran, and Y. Yu. Q2Semantic: a lightweight keyword interface to semantic search. In *ESWC*, pages 584–598, 2008.

[12] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan. Summarizing answer graphs induced by keyword queries. *Proc. VLDB Endowment.*, 6(14):1774–1785, 2013.

[13] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *VLDB Endow.*, pages 565–576, 2014.

[14] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu. Spark: Adapting keyword query to semantic search. In *ISWC*, pages 694–707, 2007.