SPARQL Query Generation based on RDF Graph

Mohamed Kharrat, Anis Jedidi and Faiez Gargouri
MIRACL LAB, University of Sfax, Sfax, Tunisia

Keywords: SPARQL, Queries, RDF.

Abstract: Data retrieval is becoming more difficult due to the heterogeneity and the huge amount of Data flowing in the Web. On the other hand, novice users could not handle querying languages (e.g., SPARQL) or knowledge based techniques. To simplify querying process, we introduce in this paper, a proposal of automatic SPARQL query generation based on user-supplied keywords. The construction of a SPARQL query is based on the top relevant RDF sub-graph, selected from our RDF Triplestore. This latter rely on our defined semantic network and on our Contextual Schema both published in two different papers of our previous studies. We evaluate 50 queries by using three measures. Results show an F-Score of about 50%. This proposal is already implemented as a web interface and the whole queries interpretation and processing is performed over this interface.

1 INTRODUCTION

The huge amount of data flowing in the Web is making research harder for end users. In the other hand, existing query languages and data models, both are hard to understand for novice users. Therefore, it's become a necessity to simplify retrieval process. Keyword search is one of available solutions enabling users expressing their requirements. Expect that Keywords based queries are usually ambiguous to interpret. On the contrary, we can easily and clearly write a SPARQL queries.

We outline in this paper a procedure for automatic SPARQL query generation based on user keywords.

After a linguistic processing of keywords, we create and select the more relevant query matching the user keywords through the construction of a set of suitable graphs acting like patterns for queries.

Our RDF TripleStore is the outcome of applying the translation process from our XML dataset using a small semantic network (Kharrat, 2015). All the querying process is based on this latter.

This proposal, is part of our retrieval solution composed by a unique search box which provides access to the whole dataset. Each result is illustrated by a node represented a physical resource (video, image, text, audio) and are linked together with other (semantically related) resources or related to another matched triples. If the user clicks on one of those nodes, a window appears, leading to details of this latter (details are specific to the type of resource result: description, playing video or audio….)

Triples are linked together, constructing a connected graph with typed links. An RDF triple is structured as SPO (Subject, Predicate, Object). Subject and object can be a simple or component noun. Theoretically, by using a keywords search, it is not possible to make a complete mapping of keywords query to a triple. Because the keywords used un a query could be ambiguous or do not match any of triples elements. In addition, there are hidden relationships between resources that could not be expressed explicitly by keywords. Keywords used in search engine for retrieving information can match to nodes or edges of a subgraph. However, many other nodes and edges could be selected as result since they have relationships with other results. Selecting this type of results is called induction.

This paper is organized as follows: In section II we present an overview of some studies along with our proposed approach of SPARQL query generation. The subsequent section III introduces the first step of our approach. In section IV we describe the process of automatic query generating based on graph patterns. We described a brief evaluation of this approach in section V. Conclusion will be finally found in last section.

Kharrat, M., Jedidi, A. and Gargouri, F.
SPARQL Query Generation based on RDF Graph.
DOI: 10.5220/0006091904500455
In *Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2016) - Volume 1: KDIR*, pages 450-455
ISBN: 978-989-758-203-5

2 RELATED WORK

Actual querying languages are largely used nowadays. However, their biggest drawback is their complexity of manipulation which make them intended for experts only.

It is obvious that there are other alternatives for novice users to manipulate RDF data. Automatic query generation is one of them which has been integrated many times in certain solutions. In the following, we present recent works about SPARQL query generation.

In (Shekarpour; 2013), author introduce a novel approach of question answering system, which transforms user keywords or sentences into conjunctive SPARQL queries. For query generation from user keywords, he employs a hidden Markov model for resources disambiguation and he use inference for the creation of a graph corresponding to SPARQL query.

Authors of (Wollbrett, 2013) and (Lehmann, 2011) have creates and annotated RDF views that enable automatic generation of SPARQL queries. Another way to generate queries is proposed in the work of Haag (Haag, 2015) who proposed a graphical manner to construct a complex queries using logical symbols. The proposed approach of (Acosta, 2015) aims first to design a model for estimating RDF completeness and then the creation of a query planner able to decide which parts of a SPARQL query will be executed. Finally, (Zhao, 2014) has presented a different approach for SPARQL queries generation based on profiling representing abstracts of heterogeneous data.

Our proposal is different from those approaches since they could not interpret implicit meaning of keywords and handle semantic relationships existing between queried data resources. In fact, in our proposal we consider two major elements: our semantic network and our Contextual Schema (Kharrat, 2015).

3 QUERYING PROCESS AND PREPROCESSING

Currently, Keywords based research is the most used for data retrieval. In the other hand, there is a huge amount of available RDF data which are getting more complex. Using SPARQL queries over them is the most adequate solution but not for novice users. Many studies for generating automatically this type of queries have been carried out. However, this kind of solutions is implying ambiguity when interpreting user's intention. To avoid as possible this problem before generating SPARQL queries we process user's keywords as a first step of our proposal.

To make a disambiguation of keywords, we compute a relatedness score of used terms to get the shared context of research and the aim of each one of them. We used Wordnet Web service which offer free linguistic processing (Grammar detection, Synonyms, proximity relations…) of any introduced term.

Taking as example the keyword « Write », Wordnet, return all derived terms of this latter (written, writing, ….). In addition, we detect Named Entities (NE) to avoid treating composed NE as separated terms and to avoid processing them as verbs, nouns or adjectives.

∀ Keyword k ∈ Query q, we define a set of derived terms of k as LE ={x1, …, xn}.

We then construct another representative set for NE called "EE" to increase the chance to find closed concept to seeking keywords.

"EE" set is more generic than 'synsets' (sets of synonyms within dictionary-oriented terms). Except that is not intended for all keywords. "EE" is a set of derived concepts of keywords in DBpedia.

This set return APk= { r1,r2,…, rn } as APk ⊂ (C ∪ I ∪ P) where C, I and P, are respectively Classes, Instances and Properties from DBpedia containing k as label.

∀ r i ϵ APk , we derive resources semantically related through these properties: rdfs:seeAlso, owl:sameAs skos:narrower and skos:narrowMatch, skos:closeMatch, skos:mappingRelation and skos:exactMatch. Notice that in DBpedia, categories are related to their top-level-categories through skos:broader and support dcterms:subject rather than skos:subject.

Taking the famous example of « Apple », returned terms from DBpedia could be : dbpedia-owl:apple_(Company), dbpedia-owl:Apple_(Fruit), dbpedia-owl:Apple_(Band)…

Using this set, we are able to eliminate ambiguity of keywords and detect shared context of them by computing relatedness score as follow:

$$sr(k1,k2) = \frac{\mathrm{Log}(\mathrm{Max}(|A|,|B|)) - \log(|A \cap B|)}{log(|W|) - log(min(|A|,|B|))} \quad (1)$$

Where k1 and k2 are two keywords, A and B are sets of derived terms respectively of k1 and k2. W is the set of all terms.

4 QUERY GENERATION PROCESS

4.1 Formal Representation

Answering a semantic query could be formulated as retrieval of interconnected objects through relations and restrictions. Figure 1 represent a view of a sample set of RDF data under two different formats. (a) triples and graphs(b). A query could be written in the following formal representation.

The RDF data set is defined as: D: {C,I,L,R,t} where C & R represent the set of Classes and relationships. I and L are respectively the set of Instances and literals.

A function T : {C ∪ I} x{C ∪ I ∪ L}→R defines all triples in D. in addition we define {e}: {C ∪ R ∪ I ∪ L} to represent all Classes, relationships, instances and literals. A keyword K is a set of terms {t}.

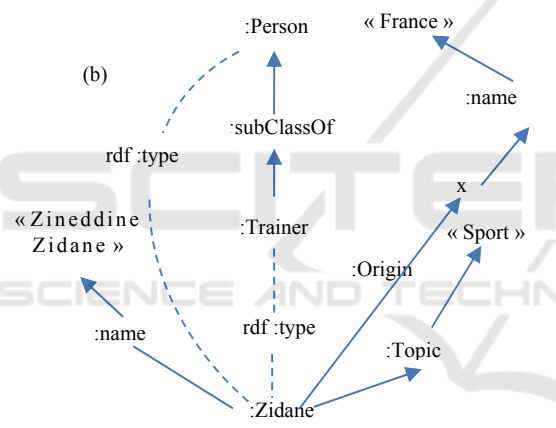

Figure 1: Sample RDF set represented as triples (a) and as a graph (b).

Formally, we represent a query as : RF : ‹C', R', I', L', V, T'› under D, a set of variables gatherings relationships and nodes.

R':‹I' ∪ C' ∪ V› x ‹ I' ∪ C' ∪ V ∪ L'› →R define all triples in RF.

Considering this definition, the problem of constructing a query could be designed as follow:

Taking a graph D, typing a keyword K, we construct queries {RF}.

To do construct s a SPARQL query covering user's keywords, we have to apply a sequel of steps to construct a graph acting as pattern. For every keyword ki ∈ K, we look for all matched Ci inside TripleStore. We then look for contexts cx inside graph as cxi ⊂ {cx} ∈ Si. Contexts are already defined inside our semantic network.

Moreover, we calculate all possible combination W for matching Q of those keywords: W = Q1 × ... × Qn = {w = (q1,...,qn)|qi ∈ Qi, ∀i = 1,...,n}. User could type one or many keywords. In case of three words for example, system may return results by combining two of them. For every combination w = (q1,...,qn) ∈ W which contains only one matched element qi per keyword ki, we generate a graph D letting us constructing it corresponding SPARQL query. Furthermore, we eliminate from graph, all elements which have low value in their uncertain property: UN. Finally, we select automatically Storylink triples: Sln ignoring contexts of the latter. Those triples do not obviously represent information seeked by user, however, they are certainly related to the same storytelling forming part of the chronological series of events. Those steps are depicted in Figure 3.

The graph D, which acting like a pattern for SPARQL query generation is defined as follow:

D=(S,A, Π) where S is a finite set of vertices defined as Cartesian product of Se ∪ Sc ∪ Sl= ({0} x Ss) ∪ ({0} x Sc) ∪ ({0} x Sl), where Ss is the set of vertices representing entities, Sc is the set of vertices representing classes and finally Sl is a set of vertices representing literals.

- A ⊆ S × S is a set of finite pairs (s',s'') edges, where s', s'' ∈ S

Π is the projection of a vertices on edge

- Π: {S U A} → α is a function which assign ∀s ∈ S and ∀ (s1,s2) ∈ A, a label of α alphabet defined as Cartesian product Ee U Es U Ec U Er U Ei. Where Ee is a set of labels for entities (applied only if e ∈ Se), Es is the set of labels for literals (applied only if e ∈ Sl), Ec is a set of labels for classes (applied only if e ∈ Sc), Er is a set of labels for properties between entities (applied only if e1 e' and e'' ∈ Se), finally, Ei is a set of labels for properties between an attribute and an entity (applied only if e' ∈ Sa & e'' ∈ Sl)

Let's take the set ENS (q1,...,qn) ∈ W, graph is constructed as following:

-if qi ∈ Es, add an edge (w, sqi) and a friend vertice with D (sqi) = qi. The latest inserted edge (w, qi) will be related to the exact w ∈ Sc, labeled

D(w, sqi) = Π(a), according to information given by term index of wq.

-if qi ∈ ΠA then we add an edge (c,s) with an edge s. Inserted edge will be related exactly to w ∈ Sc and labeled as D(w, s) = Π(w) ∈ Ei.

-if qi ∈ Er / Qi ∈ Ec then we verify if ∃ qi=Sln. If yes, then we add an edge (w,s) with a vertice s and labeled D (w,s) = Sln.

-∀qi ∈ Er, if ∃ property ∈ UN, then eliminate triple from graph.

The graph which will be kept, is the less connected of matched elements qi ∈ c with the lowest score SCO. SCO is defined as average distance between matched elements pairs qi. SCO = ∑ distance (qi, qj) ∀i, j = 1, ..., n and i ≠ j. To reduce SCO, we look for the shortest path in graph D. However, by calculating this shortest path, in some cases, qi may be an edge, so the distance between two matched elements is computed including edges and vertices leading to qi. Finally, we combine all shortest distances forming a connected sub-graph. To find relevant graph, we keep only the lowest scores.

We can, at this level, generate a SPARQL query basing on the kept graph. Prior to this, we have to apply preprocessing of keywords, as result of preprocessing, we may add a derived terms to originals ones. The entire terms and keywords will be submitted for querying.

4.2 Query Construction

We generate SPARQL queries for every graph while considering:

-if (s,o) ∈ A, where s & o ∈ Ss, L(o) ∈ Es & Π(s,o) ∈ Ei, then we generate the skeleton of the following triple: var(s) Π(s,o) var (o) FILTER (var(o)=Sln) (For Storylink triples)

-∀ (s,o) ∈ A, where s & o ∈ Ss and Π(s,o)=cx, then we generate the skeleton of the following triple: var(s) Π(s,o) var (o) FILTER (var(o)= (cxi,.. cxn) ∈ {cx}) (For Context triples)

For all the 6 semantic relationships defined in our Contextual Schema:

-if s ∈ Sc then we associate the vertice s with a new variable var(s) and generate the skeleton of the following triple: var(s) rdf :type Π(s) where E(s) ∈ Ec.

-if (s,o) ∈ A, from vertice s ∈ Sc to vertice o ∈ Sc (inter-entities property) where Π(s,o) ∈Er, then we generate the skeleton of the following triple: var(s) L(e) var(o)

-if (s,o) ∈ A, from vertice s ∈ Sc to vertice o ∈ Ss (entity-attribute property) where L(o) ∈ Es & Π(s,o) ∈ Ei, then we generate the skeleton of the following triple: var(s) Π(s,o) var (o) FILTER (var(o)=L(o)).

-if (s,o) ∈ A, from vertice s ∈ Sc to vertice o ∈ Ss (entity-attribute property) where Π(s,o) ∈ Ei, then : we generate the skeleton of the following triple: var(s) Π(s,o) var (o).

Last step before query generation is adding more restrictions about shared context of keywords and about uncertainty property.

5 EVALUATION

We present in this section, a brief evaluation of this work.

First of all, we have to make distinguish between different results. In fact, they are divided into two groups. Results including exact user's terms and results including and considering user intents. This implies the existence of correct results and those entirely relevant. For this reason, we distinct number of keywords best matching to elements from Triplestrore in order to find the best sub-graph considering all user requirements. In addition, we have to distinct correctness degree of results. For this, we introduce a measures already used in this kind of evaluation (Shekarpour; 2013).

Let's take the same example described previously. Generated query may return (correct, partially correct, or wrong) results. User keywords could match exactly with resources but those latter may not be the intent of user in terms of meaning. On that view, results may deviate towards resources just talking about Hillary in other context.

We introduce those three measures:

Correctness Rate (CR) : This measure will let us assign a score to results matching to the user's expectations of his query q. CRq(α) is the fraction of correct terms over all terms.

$$CR_q(\alpha) = \frac{Correct\ terms}{Set\ of\ terms} \quad (2)$$

(Average CR): It represents arithmetic of CR of their individual answers A for query q.

$$ACR_q(A) = \frac{1}{|A|} * \sum_{\alpha \in A} CR_q(\alpha) \quad (3)$$

Fuzzy precision (FP): The ACR is the basis of this measure, which measures the overall correctness of graphs matching answers Aq.

The ACR is the basis for the fuzzy precision metric (FP), which measures the overall correctness of a template's corresponding answers Aq with respect to a set of keyword queries Q.

$$FP = \frac{\sum_{q \in Q} ACR_q(A_q)}{|Queries\ returning\ results|} \quad (4)$$

After computing the fuzzy precision, we can evaluate the quality of the results achieved by graph patterns. Afterwards, we compute the Recall corresponding to fraction of queries which have got results.

$$Recall\ R = \frac{Queries\ returning\ results}{Set\ of\ all\ queries} \quad (5)$$

Finally, we compute F-Score as follow:

$$F = 2 * \frac{FP*R}{FP+R} \quad (6)$$

We evaluate 50 queries by using these three measures. Table 1 represents a small set of used queries for evaluation.

Table1: Sample set of used queries for evaluation.

Queries
Q1:Hillary+Isis
Q2:Hillary+Isis+Iraq
Q3:Hillary+visit+Iraq
Q4:Hillary+meeting+G8
Q5:Hillary+meeting+Obama
Q6:europe+Isis+war
Q7:Europe+Goodluck Jonathan+boko haram
Q8:Hollande+Goodluck Jonathan+boko haram
Q9:Hollande+Iraq+war+decisions
Q10:Angela Merkel+G8+decisions

Figure 2 shows the accuracy of each query based on these three metrics.

Basing on these three measures, overall F-Precision and recall values as well as an overall F-Score value were computed as the average mean of the F-Precision, recall and F-Score values for all queries. This preliminary evaluation shows an average of 0.52 of F-Score, 0.43 of Recall and 0.5 of F-Precision.

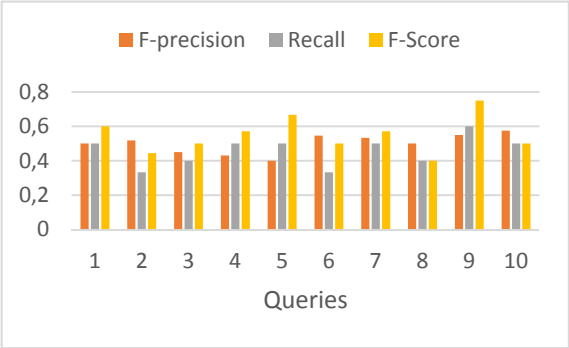

Figure 2: Accuracy of each query in Table 1.

6 CONCLUSION

In this research, we proposed an automated system for SPARQL query generation from user keywords. The process includes three main parts: Preprocessing of keywords, detecting best graph matching to keywords and finally the query generation. The aim of this proposal is to simplify data retrieval for novice users with respect of their intents, in fact of ambiguous aspects of keywords based search. In the last section we showed a brief evaluation of the proposed approach. We aim to apply some optimizations to our approach, since we currently use a keywords based Web interface.

REFERENCES

Acosta, M., Simperl, E., Flöck, F., Vidal, M., Studer, R., 2015. RDF-Hunter: Automatically Crowdsourcing the Execution of Queries Against RDF Data Sets. *Journal: The Computing Research Repository (CoRR).*

Haag, F., Lohmann, S., Siek, S., Ertl, T., 2015. QueryVOWL: Visual Composition of SPARQL Queries. In *Proceedings of Extended Semantic Web Conference-Satellite Events. Slovenia.*

Kharrat, M., Jedidi, A., Gargouri, F., 2015. A semantic approach for transforming XML data to RDF triples. *IEEE/ACIS 14th International Conference on Computer and Information Science (ICIS). USA.*

Kharrat, M., Jedidi, A., Gargouri, F., 2015. Defining Semantic Relationships to Capitalize Content of Multimedia Resources. *IFIP Advances in Information and Communication Technology, vol 456.*

Lehmann, J., Buhmann, L., 2011. Autosparql: Let users query your knowledge base. In *Proceedings of Extended Semantic Web Conference. Greece.*

Shekarpour, S., et al., Generating SPARQL queries using templates. 2013. *Journal of Web Intelligence and Agent Systems, vol 11, pp. 283-295.*

Wollbrett, J., Larmande, P., De Lamotte, F., Ruiz, M., 2013. Clever generation of rich SPARQL queries from annotated relational schema: application to Semantic Web Service creation for biological databases. *Journal Bmc Bioinformatics, vol 14.*

Zhao, J., HongHan,W., Pan, JZ., 2014. Towards Query Generation for PROV-O Data. In *Proceedings of Provenance Analytics at ProvWeek. Germany.*

APPENDIX

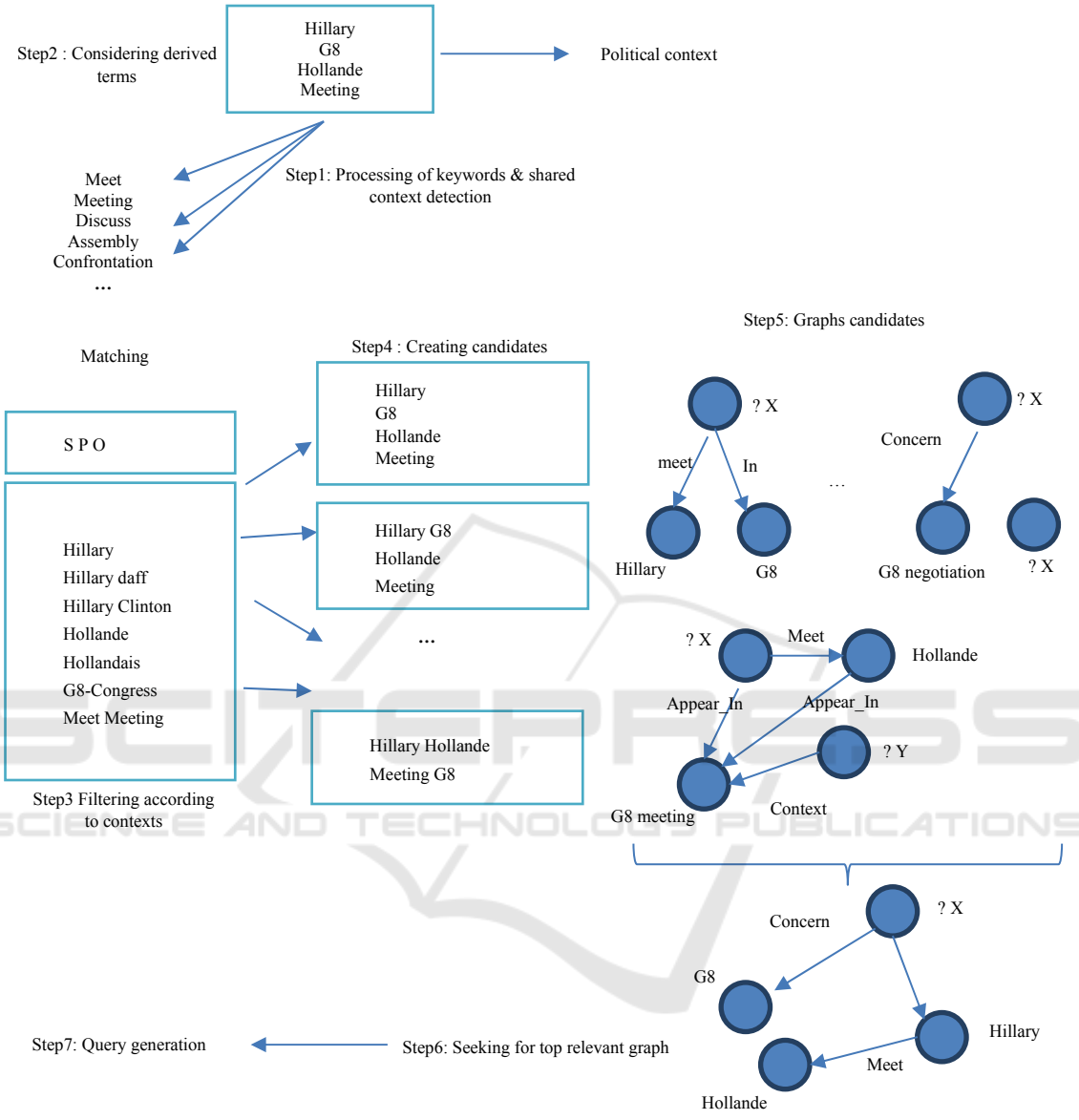

Figure 3: Processing steps of generating SPARQL queries.