# Keyword Search Over RDF Graphs Using WordNet

Mohamad Rihany
*DAVID Lab*
*Université de Versailles*
*Saint-Quentin-en-Yvelines*
Versailles, France
mohamad.rihany@uvsq.fr

Zoubida Kedad
*DAVID Lab*
*Université de Versailles*
*Saint-Quentin-en-Yvelines*
Versailles, France
zoubida.kedad@uvsq.fr

Stéphane Lopes
*DAVID Lab*
*Université de Versailles*
*Saint-Quentin-en-Yvelines*
Versailles, France
stephane.lopes@uvsq.fr

***Abstract*—An increasing amount of interlinked RDF datasets are published on the Web. These datasets can be queried using languages such as Sparql, in which graph patterns are evaluated against the data. In such languages, some knowledge about the dataset is required in order to formulate a query, such as the resources, types or properties existing in the dataset. An alternative way of querying RDF data is keyword search, which could be very useful when the content of the dataset is not known. One of the problems we are faced with is the gap between the keywords of the query and the terms used in the dataset. In this paper, we introduce a keyword search approach for RDF data which aims at solving this terminological gap. Our approach makes use of external knowledge provided by online linguistic resources and proposes a ranking method if several results are returned for a given query. We have performed some experiments on the DBpedia and the AIFB datasets to illustrate the scalability and efficiency of our approach.**

## I. INTRODUCTION

There is an increasing amount of RDF [1] datasets published on the Web, enabling knowledge extraction for numerous applications. An RDF dataset could be viewed as a graph where nodes are resources or litterals and where labeled edges represent properties. In RDF, the building block is a triple, which is of the form (subject, predicate, object). The RDF Schema (RDFS) language is used to introduce useful semantics to RDF triples. It provides a built-in vocabulary for asserting user defined schemas within the RDF model. This vocabulary can be used to specify URIs as being of a specific type (classes, properties and instances), to denote special relationships between URIs. The flexibility of the RDF data model allows the representation of both schema and instance information in the form of RDF triples.

RDF datasets can be queried using languages such as the SPARQL language where queries are specified as graph patterns evaluated against the dataset. A SPARQL query consists of a set of triples where the subject, predicate and/or object can consist of variables. The idea is to match the triples in the SPARQL query with the existing RDF triples and find solutions to the variables. In order to formulate these queries, some knowledge about the dataset is therefore required. This could be related to the subject, which could either be a specific resource, or a class existing in the dataset, or to a predicate, which represents a property describing either a class or a resource. The user should also be familiar with the SPARQL query language.

An alternative way of querying RDF dataset is keyword search, which consists in formulating a query as a set of keywords and extracting the subgraphs corresponding to the input keywords. Keyword search over RDF datasets raises several challenges. One of them is finding the relevant elements by matching the keyword query with the elements of the datasets, taking into account differences of terminologies which may exist between them. Another challenge is to aggregate the relevant elements and to built the subgraphs representing possible answers to the initial query.

Let's consider the RDF data graph about movies given in figure 1.

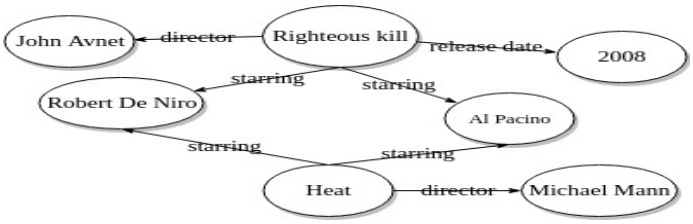


Fig. 1: RDF graph dataset

Let Q={2008, performing, film-maker} be a keyword query. If a user issues this keyword query on the data graph of figure 1, then the answer will be empty. However, knowing that "performing" is similar to "starring" and that "film-maker" is similar to "director", we can see that there is in the graph some data that is relevant to the query. This result could be retrieved if we could bridge the terminological gap between the terms of the query and the ones used in the dataset.

In this paper, we introduce a keyword search approach which takes keywords as input and returns the best matching subgraphs as an answer to this query. The key contribution is to use an external source of knowledge providing semantic relations in order to find the elements in the dataset that match the query keywords, and to rank the set of possible answers using a ranking method based on the semantic relations which have been used during the matching process.

*PhD is funded by CNRS-L and ANR (project CAIR)

The rest of this paper is organized as follows. The approach overview is provided in section 2. We detail our solution for matching keywords with graph elements using external knowledge in Section 3. Section 4 presents the process of building the end results from the matching elements.The ranking method is discussed in section 5. Section 6 presents our experiments and section 7 reviews the related works. Finally, we conclude the paper and present some future works in Section 8.

## II. APPROACH OVERVIEW

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

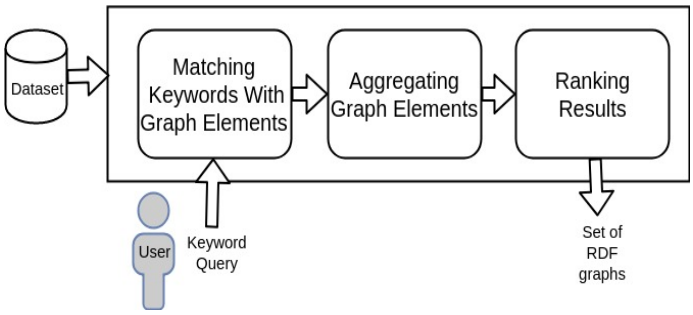


Fig. 2: Approach overview

Our goal is to provide a keyword search approach on RDF datasets, which is an alternative way to query these datasets. In our approach, the user will enter the keywords composing the query as an input and get a set of RDF graphs corresponding to the keywords as an output; but the query is not always expressed using the same terms as the ones used in the dataset. Our goal is to bridge the gap between the keywords and the dataset terminology. We propose the use of an external knowledge source providing semantic relations between concepts. In our work, we have used the Wordnet online linguistic dictionary.

Figure 2 shows our framework for keyword search. It comprises three components: matching, aggregating and ranking.

- **Matching keywords**
  The matching component takes as input the keyword query and searches for each keyword the matching elements in the dataset. This is done by comparing the keyword to each graph element (resource, class or property) and returning the matching ones. In some cases, the user may enter a keyword for which an exact match can not be found in the dataset, but some graph element could be close to the keyword, it could for example be a synonym, or a close concept. The problem is to identify the equivalent concepts and the close concepts to some keyword in the dataset. To do so, we suggest the use of external knowledge sources such as online linguistic dictionnaries.

- **Aggregating Graph Elements**
  Once the matching elements in the RDF graph are identified for each keyword, the problem is to built the final result from these elements, and to aggregate them into a connected subgraph representing an answer to the query. Each keyword can be associated to more than one element in the RDF graph; we consider that each combination of matching elements where there is one element for each keyword is a possible answer to the query. The problem is to built the subgraph containing the elements of the considered combination.

- **Result Ranking**
  As each keyword may have more than one matching element in the dataset, there may be several possible results to the query. The problem is to rank the different results and to find a ranking method capable of identying the results that are closer to the initial query than others.

## III. MATCHING KEYWORDS WITH GRAPH ELEMENTS

In this section, we present our approach for matching the keywords with the data graph using external knowledge. Let the keyword query Q=$\{k_1, k_2, k_3, ...k_n\}$ be the input. For each keyword $k_i$ we check if there exists an exact match in the elements of the data graph or not. If such element does not exist, we search the considered external knowledge source in order to find some semantic relations between a keyword and a concept such that this latter concept has a matching element in the data graph. At the end, we obtain for each $k_i$ a set of matching elements from the data graph.

For example, let us consider the keyword query Q={2008, performing, film-maker}. We can observe from figure 1 that the keywords "performing" and "film-maker" have no matching elements, but if some external knowledge source is available and provides us with the information that "performing" is similar to "starring" and "film-maker" is similar to "director", then some matching elements for the keywords of Q can be found, and they are presented in figure 3.

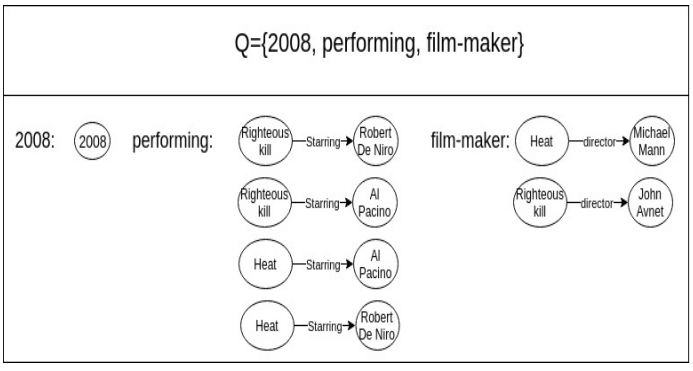


Fig. 3: Matching Elements for Each Keyword in the Query Q

### A. Matching using external knowledge

In our work, we have used WordNet as an external knowledge source. WordNet is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each one expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. This lexical database can help to find the best matching between the keyword queries and the dataset elements using the provided semantic relations (Synonyms, Antonyms...). We have decomposed the matching keywords process into two phases. The first one is the search in the dataset for exact matches for a given keyword. The second phase is the search for close matching elements in the dataset.

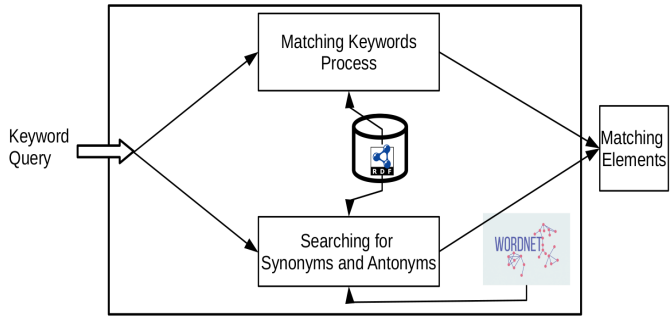


Fig. 4: Searching for Exact Matching Elements

In figure 4 we have represented the first phase, which is the search for exact matches int the dataset. Consider a keyword query Q=$\{k_1, k_2, k_3, ...k_n\}$. For each keyword $k_i$ we perform two searching tasks in parallel; the first one is searching the dataset for properties, resources or classes that have the same name as the considered keyword, i.e., nodes and edges in the data graph having $k_i$ as label. The second task is to search the knowledge base for synonyms and antonyms of the considered keyword. This consists in querying WordNet to extract the semantic relations (synonyms antonyms and hypernyms) with $k_i$ and find the exact matching of those semantic relations in the data graph.

If the search for exact matching elements fails, then a search for close elements is performed. If some matching element is found in the dataset for every keyword, then the search process ends.

In figure 5 we have represented the search for close matching elements. This process will be executed for a given keyword only if no exact matching element has been found. It is similar to the search for exact matches and differs only on the considered semantic relations in Wordnet. The semantic relations we are interested in are hyponymy, holonymy and meronymy. For each keyword $k_i$ which has no exact matching element in the dataset, we query the WordNet database to search for one of the previous semantic relations.

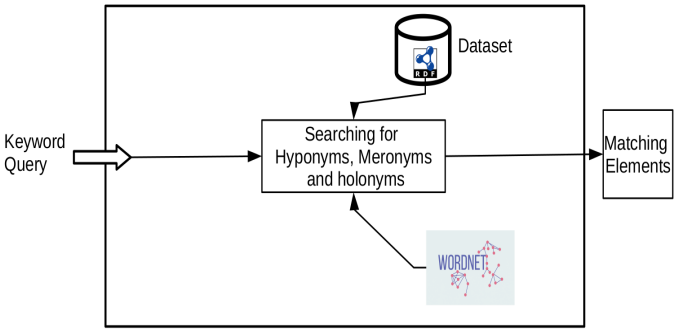


Fig. 5: Searching for Close Matching Elements

These relations do not express equivalence, but express some sort of closeness between two concepts. For example, if a meronymy relation is found between $k_i$ and a concept c, and if c has an exact match in the dataset, i.e. a graph element labeled c, then this latter is a close concept to $k_i$. Indeed, if c does not represent an equivalent concept, it still represents a close concept as c is part-of $k_i$ because the two are linked by a meronymy relation according to Wordnet.

Close matching elements are searched for each keyword without exact matching element. If some keyword in the query has neither exact matching elements nor close matching elements, this means that there is no answer to the query.

There are many semantic relations in WordNet, some of them are very useful to help us find the best matching elements between the query and the dataset. In our work, we have considered the following relations to search for matching elements.

- Synonyms: a concept that means exactly the same as another.
- Antonym: a concept opposite in meaning to another.
- Hyponym: a concept whose meaning denotes a subordinate.
- Hypernym: a concept whose meaning denotes a superordinate.
- Substance meronym: a concept that is a substance of another concept.
- Part meronym: a concept that is part of another concept.
- Member meronym: a concept that is part of another concept.
- Substance of holonym: a concept that has another concept as a substance.
- Part of holonym: a concept that has another concept as a part.
- Member of holonym: a concept that has another concept as a member.
- Cause to: a verb that is the cause of a result.
- Troponym: a verb that is particular way to do another.

*B. Matching Algorithm*

In this section, we present the matching algorithm underlying our approach, which matches the keywords of the query with the dataset. Let us start by presenting the notations used in the algorithm. Let K=$\{k_1, k_2, k_3...k_n\}$ be the keyword query, ME($k_i$) a function to extract the matching elements for $k_i$ in the dataset (these elements can be literals, instances, classes, properties, etc.).
The semantic relations between the keyword query $k_i$ and WordNet are extracted by using some functions, for example Synonym($k_i$) is used to extract the set of synonyms for the keyword query $k_i$.
For each keyword $k_i$ in the query, the matching elements ME($_i$) are extracted (line4-5), and then WordNet is queried to extract the set SR of Synonyms, Antonyms and Hyponyms. For each element of SR, the dataset is accessed to check if there is a matching element (line6-16). For example, the search for matching elements using the antonymy relation is done by issuing the following query to Wordnet.

SELECT ?y
WHERE
$\{ki\ Antonym\_to\ ?x.$
$?x\ Antonym\_to\ ?y.\}$
$And\ k_i\ different\ from\ ?y)$

If no matching element has been found, then a search for close matching elements is performed (line 17). In this phase, WordNet is queried to find close matching elements by searching for the hypernymy, holonymy and meronymy semantic relations for $k_i$. For each concept c related to $k_i$ by one of these relations, we search for elements in the dataset labeled c; these elements are added to the set of matching elements for the keyword $k_i$ (line 17-31).

## IV. Aggregating graph elements

After obtaining the set of matching graph elements for each keyword, the subgraphs representing the possible answers to the query are built. Each answer is a connected minimal subgraph which contains for each keyword $k_i$ one corresponding matching element.
Considering that there is a set of matching elements for each keyword, we first derive the possible combinations by computing the cartesian product of the different sets of matching elements. Then, for each combination, we determine the minimal connected subgraph containing the matching elements of the considered combination. Each subgraph is a possible answer to the initial query. Let us consider the example of the keyword query given in Figure 3, with the set of matching elements corresponding to each keyword. In figure 6, we can see all the possible combinations, obtained by performing the cartesian product of the sets of matching elements shown in figure 3.

From each combination, a connected subgraph will be extracted. This subgraph represents a possible result to the query; it is built by introducing the minimum number of nodes

**Algorithm** Matching The Keywords Of The Query With The Graph

1: $keywordQuery$ = $\{k1, k2, k3, ...ki\}$
2: **procedure** MATCHING(keywordQuery)
3: $hashmap(keyword, relatedElements)$
4: **for** each keyword $ki$ in $keywordQuery$ **do**
5: ME($ki$) ▷ extract the matching elements for the keyword query $ki$
6: $S \leftarrow synonym(ki)$ ▷ $S$ is set of synonyms to the keyword $ki$
7: **for** each $sj$ in $S$ **do**
8: $ME$(ki) $\leftarrow ME$(ki) + $ME$(sj) ▷ extract the matching elements for the synonym $sj$ and add them as matching element to the keyword query $ki$
9: **end for**
10: $a \leftarrow Antonym(ki)$ ▷ function Antonym take ki as input and return keyword similar to it as output after executing the following query {ki Antonym to ?x, ?x antonym to ?y.} and ki different from ?y
11: $ME(ki) \leftarrow ME(ki) + ME(a)$
12: $Hyper \leftarrow Hypernym(ki)$ ▷ $Hyper$ is set of Hypernyms to the keyword $ki$
13: **for** each $hj$ in $Hyper$ **do**
14: $ME$(ki) $\leftarrow ME$(ki) + $ME$(hj) ▷ extract the matching elements for the Hypernym $hj$ and add them as matching element to the keyword query $ki$
15: **end for**
16: $hashmap.add(ki, ME(ki))$
17: **if** ME($ki$) is empty **then**
18: $Hypo \leftarrow Hyponym(ki)$ ▷ $Hypo$ is set of Hyponyms to the keyword $ki$
19: **for** each $hj$ in $Hypo$ **do**
20: $ME$(ki) $\leftarrow ME$(ki) + $ME$(hj)
21: **end for**
22: $Mero \leftarrow Meronym(ki)$ ▷ $Mero$ is set of Myronyms to the keyword $ki$
23: **for** each $mj$ in $Mero$ **do**
24: $ME$(ki) $\leftarrow ME$(ki) + $ME$(mj) ▷ extract the matching elements for the meronym $mj$ and add them as matching element to the keyword query $ki$
25: **end for**
26: $Holo \leftarrow Holonym(ki)$ ▷ $Holo$ is set of Holonym to the keyword $ki$
27: **for** each $hj$ in $M$ **do**
28: $ME$(ki) $\leftarrow ME$(ki) + $ME$(hj) ▷ extract the matching elements for the Holonym $hj$ and add them as matching element to the keyword query $ki$
29: **end for**
30: $hashmap.add(ki, ME(ki))$
31: **end if**
32: **end for**
33: Return $hashmap$
34: **end procedure**

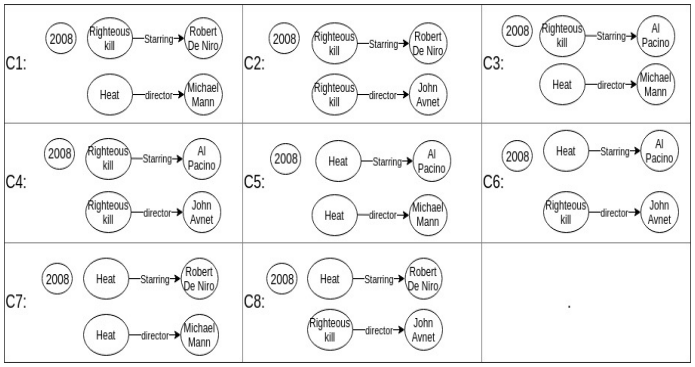


Fig. 6: Combinations of Matching Elements

which do not correspond to matching elements. To do so, we use the shortest path algorithms (Dijkstra's Algorithm).

A matching element can a be node or an edge. If we compute the shortest path between two matching elements representing both a node, then the shortest path between them is the shortest path between the corresponding two nodes in the data graph. If one of the matching elements is an edge between nodes n1 and n2 and the other matching element is a node n, then the shortest path is determined between n and one of the two nodes n1 and n2.

If the two matching elements are both edges, then the shortest path is determined between one of the nodes connected by the first matching element to one of the nodes connected by the second matching element.

For each combination, our algorithm for extracting a subgraph starts by randomly selecting one of the matching elements and then finds all the shortest paths from this matching element to all the other matching elements in the combination. Let us consider two matching elements $m_i$ and $m_j$.

$n_i \in m_i \; and \; n_j \in m_j \; such \; that \nexists n_{i2} \in m_i, \nexists n_{j2} \in m_j, \; n_i \neq n_{i2}, \; n_j \neq n_{j2} \; and \; |(n_{i2} - -n_{j2})| < |(n_i - -n_j)|$

where $|(x - -y)|$ 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 matching elements. When all the combinations are processed, we obtain a set of subgraphs, each one representing a possible result to the query.

## V. Ranking Results

The elicitation of all the combinations of graph elements and the aggregation of graph elements for each combination lead to several subgraphs, each one being a possible answer for the query. One problem is to rank these answers, and to determine if there are better results than others. In our approach, we ranked the results according to the matching process. Let us first define the notations used in the ranking method. Exact matching elements are the elements found during the first phase of the matching, i.e. the search for exact matching elements; approximate matching elements are the elements found during the search for close matching elements; we denote by linking elements the nodes that are in the subgraph result but are neither exact nor approximate matching elements. We calculate the ranking score as follows:

$$\text{Score} = 1 - \frac{[w_a * A + (1 - w_a) * L]}{N}$$

Where A is the number of approximate matching elements, L is the number of linking elements, N is the total number of nodes and edges in the subgraph and $w_a$ is the weight for A.

Intuitively, the above score expresses that the less linking elements in a subgraph, the better the solution. It also expresses that the more exact matching elements in a subgraph, the better the solution.

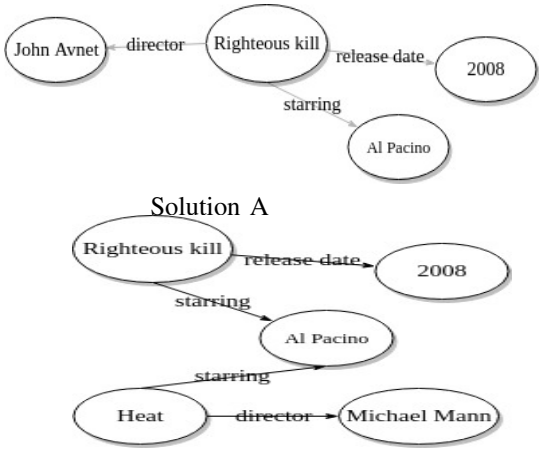


Fig. 7: Some Possible Solutions for Q

Let us consider, in our running example, the possible solutions for the query Q={2008, performing, film-maker} given in figure 7. For solution A, the number of exact matching elements is equal to one, the number of approximate matching elements is equal to two, and the number of linking elements is equal to one; the total number of nodes and edges N is equal to seven. The final score is therefore equal to 0.814. For solution B, the number of exact matching elements is equal to one, the number of approximate matching elements is equal to two, the number of linking elements is equal to two; the number N of nodes and edges is equal to nine. The final score is therefore equal to 0.778. Solution A is better than solution B. In A, the result is the movie "Righteous kill" released in "2008" directed by "John Avnet" and "Al Pacino" is one of the stars of the movie, while solution B describes the movie "Righteous kill" released in "2008" with "Al Pacino" being one of the stars, also starring in the movie "Heat" directed by "Michael Mann". Therefore solution A focuses on the release date, stars and director of one movie which is "Righteous kill", while solution B presents the release date of "Righteous kill" and the director of "Heat".

## VI. Experimental evaluation

Our approach is implemented in Java, we have used the Jena API for the manipulation of RDF data. For indexing and searching the keyword query, we have used the Lucene API. The Jung API is used for graph manipulation and visualization.

In the rest of this section, we describe our experiments to validate the performances of our approach. Our goal is to observe the performance of using WordNet as an external knowledge to fill the gap between the keywords and the dataset terminologies, as well as the ranking model with various keyword queries. All the experiments have been done on Intel Core i7 with 32GB RAM.

### A. Datasets

We have used two datasets: AIFB and DBpedia. AIFB is a dataset containing data taken from the AIFB institute, at 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 29 233 triples. DBpedia is a project aiming to extract structured content from the information created in the Wikipedia project. The extracted data is related to movies their title, stares, director, released data and other properties. This dataset contains 30 793 triples.

The size of the keyword queries was between 2 and 8 keywords. The total number of queries was 20 queries (10 for each dataset) in order to cover all the different WordNet semantic relations during the matching stage.

### B. Methodology

We have tested and compared our keyword search approach both with and without the use of external knowledge. We will refer to the approach with external knowledge by semantic approach since we use semantic relations, and we will refer to the approach without external knowledge as the basic approach.

### C. Results

The query size was between 3 and 8 keywords. Table 1 shows some examples of keyword queries, the number of nodes and edges containing this keyword in the data graph, and the semantic relations between the keywords that do not appear in the data graph and WordNet. For example, Query 1 (carole lombard-5940-theoretical-edwin) consists of 4 keywords, these keywords appear in the data graph (nodes and edges) 14, 18, 0 and 10 times respectively, this means that theoretical is not in the dataset but we can replace it with academic since there is a hypernymy relation between academic and theoretical in WordNet.

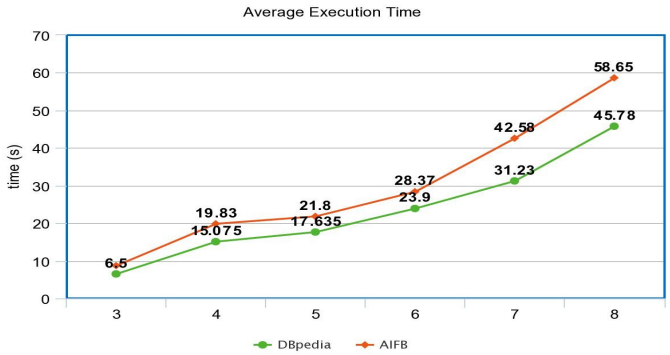


Fig. 8: Average Execution Time According to the Size of the Query

As we can observe from figure 8, the execution time increases when the number of the keywords increases for both datasets. We can also see that the execution time for AIFB is greater than the execution time of DBpedia because the size of data in AIFB is greater than the size in DBpedia.

The type and number of keyword elements also affect the execution time; for example Q1 {carol_lombard / 5940 / theoritical / edwin} in table I takes 4.91 sec (fig. 9) and contains 4 keywords while Q6 {poor / 1990 / mind / Ellen_Burstyn} needs 31.57 sec (fig. 9) to be executed: the two queries contain the same number of keywords (4), but the difference in the execution time is due to the keyword elements. In Q1 the four keywords appear 14, 18, 0 and 10 times respectively as we can see from table I but the keywords in Q6 appears 20, 258, 0 and 2 times respectively.

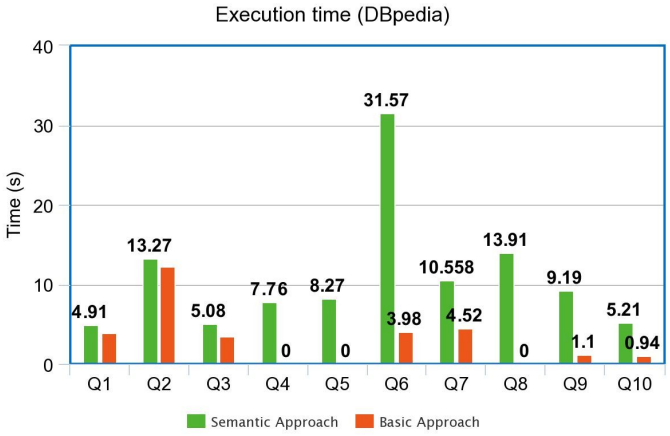


Fig. 9: Execution Time for Each Query over DBpedia

The differences in terminology between the keyword query and the dataset also affect the execution time (the keyword query does not match with any element in the dataset). Consider the queries Q3 and Q4 in table I; Q3 consists of 4 keywords, and these keywords appear 2, 1, 4 and 0 times respectively in the dataset while Q4 has 3 keywords and these keywords appear 0, 1 and 0 times respectively in the dataset.

| Query Number | keyword Query | number of appear Nodes/edges | WordNet Semantic Relations |
|---|---|---|---|
| Q1 | carole_lombard<br>5940<br>theoretical<br>edwin | 14<br>18<br>0<br>10 | hypernym |
| Q2 | swedishfilms<br>lagercrantz<br>1997<br>plosion | 2<br>4<br>97<br>0 | hyponym |
| Q3 | psychiatric<br>hampshire<br>ziskin<br>system | 2<br>1<br>4<br>0 | hyponym |
| Q4 | bring<br>cut<br>mind | 0<br>1<br>0 | antonym<br><br>hypernym |
| Q5 | solar_system<br>secondary<br>vocabulary | 0<br>0<br>0 | holonym<br>synonym<br>meronym |

TABLE I: Keyword Queries

| query | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|---|---|---|---|---|---|---|---|---|---|---|
| number of results(semantic approach) | 2 | 173 | 7 | 8 | 50 | 250 | 110 | 132 | 78 | 34 |
| number of results(basic approach) | 1 | 150 | 1 | 0 | 0 | 54 | 25 | 0 | 2 | 2 |

TABLE II: Number of Results for each Keyword Query(DBpedia)

| query | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 | Q8 | Q9 | Q10 |
|---|---|---|---|---|---|---|---|---|---|---|
| number of results(semantic approach) | 9 | 10 | 21 | 25 | 32 | 38 | 20 | 18 | 25 | 5 |
| number of results(basic approach) | 3 | 4 | 10 | 7 | 12 | 22 | 14 | 7 | 13 | 0 |

TABLE III: Number of Results for each Keyword Query(AIFB)

But the execution time for Q4 (7.76 sec) is greater than the execution time of Q3 (5.08 sec); this is because Q4 requires to access WordNet two times to search for semantic relations involving the keywords, while Q3 requires only one access.

As we can observe from tables II and III, the number of results increases when WordNet is used during keyword search, because using WordNet increases the number of matching elements. This means that the number of combinations and therefore the number of results both increase.

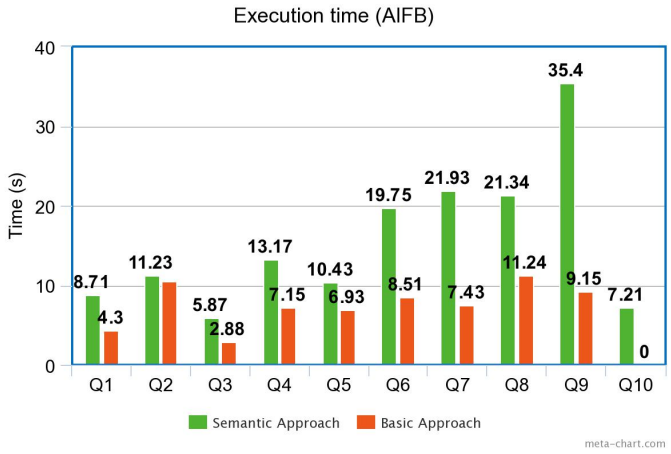


Fig. 10: Execution Time for Each Query over AIFB

To check the effectiveness of the evaluation we have used 10 queries from the tables II and III and asked three users to check the top-k results for each query and give the number of relevant results to calculate the Top-K precision according to this equation:

$$Top - kPrecision = \frac{NumberOfRelevantResults}{K} \quad (1)$$

All the results were above 0.92 as shown in table 4; this means that the results were accurate according to the users.

| Data | AIFB | | DBpedia | |
|---|---|---|---|---|
| K | 5 | 10 | 5 | 10 |
| Top-K precision | 0.97 | 0.95 | 0.98 | 0.97 |

TABLE IV: Top-K precision

## VII. Related works

Keyword search and the translation of a keyword query into a formal query have been the topic of several research works. The early research works were on keyword query over relational databases [2], [3], then XML data [4] and RDF data [6], [5], [8], [7]. One of the important problems addressed by these works is how to fill the gap between the keywords in the query and the terms used in the dataset. The SPARK approach [5] consists in finding the corresponding ontology for each term in the keyword query and try to map and find a relation between the ontology and the keyword query; other approaches use external knowledge or resources such as in the Q2semantic approach [8], where Wikipedia is used to extract related keywords; for each keyword in the dataset, a document is created containing features that are matched to

the keyword Query. The approach described in [9] also uses some external knowledge; it uses the supporting entity pairs in order to paraphrase dictionary records semantic equivalence between relation phrase and dataset; but the supporting entity pairs are specific to Wikipedia and the New York Times [10].

In order to aggragate the matching elements in a dataset, SPARK[5] uses an ontology to discover the relations between the keywords and the dataset and uses a minimal spanning tree algorithm to create a possible query graph. The approaches described in [8], [6], [11], [14], [15] transform the data graph into a summarized graph; some of them start from the leaf nodes containing the keyword query and do a traversal until all the paths converge to the same node, and the other works use the summarized graph and try to extract a SPARQL query by finding relationships between the nodes.[12] classifies the keywords into two sets: the first one contains the vertices and the second one contains the edges; then the final possible solutions are computed. In all these works, the keyword in the query are matched with the nodes of the considered graph, unlike our approach which considers semantic relations and searches for matching elements in both the nodes and the edges in order to build the final result.

## VIII. CONCLUSION AND FUTURE WORKS

In this paper, we have proposed an approach for keyword search in an RDF dataset, which represents an alternative to the use of query languages such as Sparql. We have focused on the problem of handling differences in the terminologies between the RDF dataset and the keyword query. We have proposed a novel solution which relies on an external knowledge source. In our approach, the answer to the keyword query is a set of subgraphs containing for each keyword one matching graph element. We have described how we enrich the set of solutions by using WordNet. We have also provided a ranking model to rank the resulting subgraphs. This model is based on the number of semantic relations extracted from WordNet and the number of nodes and edges for each subgraph. We have conducted some experiments which have shown that external knowledge gives more results for some queries and sometimes returns an answer where other approaches fail to.

In future works, we will study the possible improvements to the aggregation of matching elements and try to find other ways of combining them. We will also study scalability issues and enable efficient keyword search for massive datasets.

## REFERENCES


[1] https://www.w3.org/RDF/.

[2] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In VLDB, pages 670–681, 2002.

[3] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In ICDE, pages 724–735, 2009.

[4] Guo, Lin, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. "XRANK: Ranked keyword search over XML documents." In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 16-27. ACM, 2003.

[5] Zhou, Qi, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. "SPARK: adapting keyword query to semantic search." In The Semantic Web, pp. 694-707. Springer, Berlin, Heidelberg, 2007.

[6] He, Hao, Haixun Wang, Jun Yang, and Philip S. Yu. "BLINKS: ranked keyword searches on graphs." In Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 305-316. ACM, 2007.

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

[8] Wang, Haofen, Kang Zhang, Qiaoling Liu, Thanh Tran, and Yong Yu. "Q2semantic: A lightweight keyword interface to semantic search." In European Semantic Web Conference, pp. 584-598. Springer, Berlin, Heidelberg, 2008.

[9] Zou, Lei, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. "Natural language question answering over RDF: a graph data driven approach." In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, pp. 313-324. ACM, 2014.

[10] Nakashole, Ndapandula, Gerhard Weikum, and Fabian Suchanek. "PATTY: a taxonomy of relational patterns with semantic types." In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 1135-1145. Association for Computational Linguistics, 2012.

[11] Li, Feifei, Wangchao Le, Songyun Duan, and Anastasios Kementsietsidis. "Scalable keyword search on large RDF data." IEEE Transactions on Knowledge and Data Engineering 1 (2014): 1.

[12] Han, Shuo, Lei Zou, Jeffery Xu Yu, and Dongyan Zhao. "Keyword Search on RDF Graphs-A Query Graph Assembly Approach." In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, pp. 227-236. ACM, 2017.

[13] Ouksili, Hanane, Zoubida Kedad, Stéphane Lopes, and Sylvaine Nugier. "Using Patterns for Keyword Search in RDF Graphs." In EDBT/ICDT Workshops. 2017.

[14] Ayvaz, Serkan, and Mehmet Aydar. "Using RDF Summary Graph For Keyword-based Semantic Searches." arXiv preprint arXiv:1707.03602 (2017).

[15] Lin, Xiao-Qing, Zong-Min Ma, and Li Yan. "RDF Keyword Search Using a Type-based Summary." JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, no. 2 (2018): 489-504.