

AMERICAN UNIVERSITY OF BEIRUT

Keyword Search Over Weighted  
Keyword-Augmented RDF Graphs

by

Hrag Artine Yoghourdjian

A thesis

submitted in partial fulfillment of the requirements  
for the degree of Master of Science  
to the Department of Computer Science  
of the Faculty of Arts and Sciences  
at the American University of Beirut

Beirut, Lebanon  
February 2016

# AMERICAN UNIVERSITY OF BEIRUT

## Keyword Search Over Weighted Keyword-Augmented RDF Graphs

by  
Hrag Artine Yoghourdjian

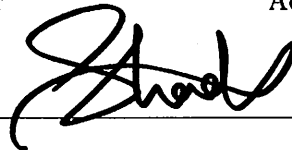
Approved by:

---

Dr. Shady Elbassuoni, Assistant Professor

Advisor

Computer Science



---

Dr. Wassim El-Hajj, Associate Professor

Member of Committee

Computer Science



---

Dr. Mohamad Jaber, Assistant Professor

Member of Committee

Computer Science



Date of thesis defense: February 5, 2016

# AMERICAN UNIVERSITY OF BEIRUT

## THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Yoghverdjian Hrag Artine  
Last First Middle

Master's Thesis       Master's Project       Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project**, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

~~Y Hrag~~  
Signature

12/Feb/2016  
Date

# Acknowledgements

I cannot find words to express my appreciation to my advisor, Dr. Shady Elbassuoni for his constant guidance, advice, and time invested for the success of this thesis.

From the very start and during each step of this work, Dr. Shady dedicated all the time needed to help me overcome the problems that I faced in completing this thesis.

I owe special thanks to Dr. Wassim El-Hajj and Dr. Mohamad Jaber for their valuable suggestions and objective criticisms during the different phases of the thesis.

I am truly grateful for the support of my family and special friends who were there for me when I needed them and greatly helped me in completing my thesis.

Finally, I would like to thank all the participants that gave their time and helped in evaluating the effectiveness of our proposed approach.

# An Abstract of the Thesis of

Hrag Artin Yoghourdjian for Master of Science  
Major: Computer Science

Title: Keyword Search Over Weighted Keyword-Augmented RDF Graphs

Large knowledge bases consisting of entities and relationships between them have become vital sources of information for many applications. Most of these knowledge bases adopt the Semantic-Web data model RDF as a representation model. Querying these knowledge bases is typically done using structured queries utilizing graph-pattern languages such as SPARQL. However, such structured queries require some expertise from users which limits the accessibility to such data sources. To overcome this, keyword search must be supported. In this thesis, we develop a retrieval model for keyword queries over RDF graphs. Our model retrieves the top-k most relevant sub-graphs to a given keyword query. To be able to do this, we augment the searched RDF graph with keywords which are extracted from entity labels and textual patterns for relations. Moreover, we associate each triple in the RDF graph with a weight reflecting the importance of the triple. Finally, we deploy a graph searching algorithm that searches this weighted keyword-augmented RDF graph and retrieves the top-k most relevant sub-graphs to the given keyword query, where relevance is measured based on the triple weights. We evaluate the effectiveness of our retrieval model using a real-world RDF dataset and compare it to various state-of-the-art approaches for keyword search over RDF graphs.

# Contents

<b>Acknowledgements</b>	<b>v</b>
<b>Abstract</b>	<b>vi</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Our approach and Contribution . . . . .	3
1.3 Overview of the Thesis . . . . .	4
<b>2 Literature Review</b>	<b>5</b>
<b>3 Our Approach</b>	<b>8</b>
3.1 System Overview . . . . .	8
3.2 Query Segmentation and Resource Matching . . . . .	9
3.3 Retrieval Model . . . . .	12
3.4 Scoring Function . . . . .	16
3.5 Termination Condition . . . . .	17
<b>4 Experiments</b>	<b>19</b>
4.1 Dataset . . . . .	19
4.2 Competitions . . . . .	20
4.2.1 Scalable Keyword Search on Large RDF data . . . . .	20
4.2.2 Web Object Retrieval . . . . .	20
4.2.3 Keyword Search Over RDF . . . . .	21
4.3 Experimental Setup . . . . .	22
4.4 Experimental Results . . . . .	23
<b>5 Conclusion and Future Work</b>	<b>26</b>
<b>A Abbreviations</b>	<b>27</b>

B All Queries	28
C Gold Queries	30
D Guidelines	31
Bibliography	33

# List of Figures

1.1	An example RDF graph about movies . . . . .	2
4.1	Average NDCG values with different alpha values . . . . .	24
4.2	Average NDCG values for four approaches . . . . .	25



# List of Tables

1.1	The set of RDF triples for the RDF graph in Figure 1.1 . . . . .	2
3.1	A Subgraph for the query "Germany England" . . . . .	16
4.1	Backward Search results . . . . .	20
4.2	Entity Based Results . . . . .	21
4.3	Backward Search results . . . . .	22
4.4	alpha = 0.3, query = "woody allen scarlett johansson" . . . . .	24
4.5	alpha = 1, query = "woody allen scarlett johansson" . . . . .	24

# Chapter 1

## Introduction

### 1.1 Motivation

The continuous growth of knowledge sharing communities like Wikipedia and the advances in automated information extraction from Web pages [1, 2] have made it possible to build large-scale knowledge bases. Examples of such knowledge bases include YAGO [3], DBpedia [4], Freebase [5], and also community-specific collections such as DBLife [6] or Libra [7]. These repositories contain entities such as people, movies, books, etc. and the relationships between them such as `bornIn`, `actedIn`, `hasGenre`, `isAuthorOf` and so on. Such data is typically represented in the form of subject-predicate-object (SPO) triples of the Semantic-Web data model RDF [8], where subjects and objects are entities and predicates are relationships between pairs of entities. An RDF collection conceptually forms a large graph, which we refer to as an RDF graph, with nodes corresponding to subjects and objects and labeled-edges denoting predicates. We refer to the entities with their relations (nodes and edges) as resources of the graph, where these resources are in the form of URIs in the dataset.

RDF is also gaining popularity in the scientific domain (e.g., in biological networks [9]), for social Web2.0 applications [10], and as a light-weight representation for the "Web of data" [11]. Overall, RDF collections can store various types of structured data. They serve as data sources that semantic-search engines can operate on to retrieve precise information that is hard to retrieve using a traditional Web-search engine. Figure 1.1 shows an example RDF graph of a movie knowledge base and Table 1.1 shows the corresponding RDF triples.

RDF data can be queried using a conjunction of *triple patterns*. A triple pattern is a triple with variables and the same variable in different patterns denotes a join condition. For example, to find "comedies that have won the Academy Award" the following 2 triple-patterns query can be used (where a conjunction is denoted by ';'): `<?x hasGenre Comedy; ?x hasWonPrize Academy_Award>` .

Given a query with n-triple patterns, the results are all the subgraphs with n

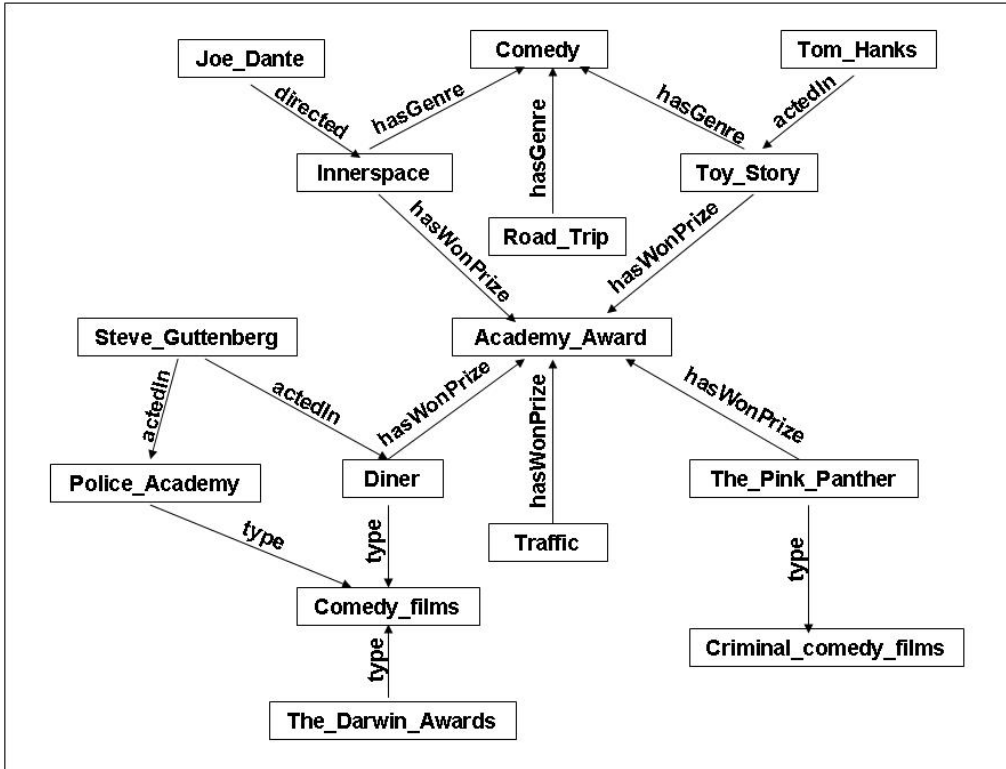


Figure 1.1: An example RDF graph about movies

Subject (S)	Property (P)	Object (O)
Traffic	hasWonPrize	Academy_Award
Innerspace	hasWonPrize	Academy_Award
Innerspace	hasGenre	Comedy
Joe_Dante	directed	Innerspace
Toy_Story	hasWonPrize	Academy_Award
Road_Trip	hasGenre	Comedy
Toy_Story	hasGenre	Comedy
Tom_Hanks	actedIn	Toy_Story
Diner	hasWonPrize	Academy_Award
Diner	type	Comedy_films
Steve_Guttenberg	actedIn	Diner
The_Pink_Panther	type	Criminal_comedy_films
The_Pink_Panther	hasWonPrize	Academy_Award
Police_Academy	type	Comedy_films
Steve_Guttenberg	actedIn	Police_Academy
The_Darwin_Awards	type	Comedy_films

Table 1.1: The set of RDF triples for the RDF graph in Figure 1.1

triples that are isomorphic to the query when binding the query variables with matching entities and relations in the knowledge base. For example, a result for the example query above could be the following subgraph: `<Innerspace hasGenre Comedy; Innerspace hasWonPrize Academy_Award>` .

Even though structured queries like the one above, allow users to represent their information needs very precisely; they are also very restrictive. They require the users to be familiar with the underlying data and a structured-query language like SPARQL. Empowering users to search RDF graphs using keywords only can increase the usability of such data sources. In addition, it enables adapting the state-of-the-art IR searching and ranking techniques.

## 1.2 Our approach and Contribution

In this thesis, we tackle the problem of keyword-query-processing over RDF knowledge bases. In particular, we implement a set of algorithms to process keyword-queries and retrieve the top- $k$  highest scored results. The results are then scored with a ranking model further explained in Chapter 3.

Our main contributions can be summarized as follows.

- We develop a suit of algorithms that process keyword queries over weighted keyword-augmented RDF graphs and return a set of RDF subgraphs matching the query.
- We develop a ranking model for keyword queries over weighted keyword-augmented RDF graphs. Our ranking model should be theoretically-founded and should take into consideration the weights of the triples.
- In order to efficiently process keyword queries over large weighted keyword-augmented RDF graphs, we develop a set of top-k query processing algorithms. Our algorithms should combine the retrieval with ranking and provide guarantees that for any given  $k$ , the returned results are indeed the  $k$  highest ranked ones.
- Finally, we test our algorithms and models on a very large real-world RDF graph, YAGO which contains millions of triples. Moreover, we plan to implement a real keyword search engine over this dataset and make it publicly available.

As an example, consider running the query "comedy academy award" against the RDF graph in Figure 1.1. A result to such a query could be the subgraph `<Innerspace hasGenre Comedy; Innerspace hasWonPrize Academy_Award>` . Note that our goal is not to return entities as results to a given query. We believe that subgraphs provide more concise answers to the user's information need, since they provide in addition to entities, the relationships between them.

Also note that our aim is not to limit results to tree-structured ones only. For instance, a result to the query "woody allen director actor" would be the subgraph `<Woody_Allen directed Manhattan; Woody_Allen actedIn Manhattan>` which is not a tree.

To retrieve a set of subgraphs given a keyword query as done in the above example, we perform the following steps. First, we need to process the query: identify and differentiate the relations (predicates) from entities by matching them to their resources. Second, for each entity from the query we expand its neighbors. Third, we continue by expanding the neighbors of the entities while keeping the complete path with the weight, meaning  $\{entity, neighbor, \dots, neighbor_n, w(p)\}$ . Fourth, we expand the vertices containing the lowest weight - when a path contains all the matching URIs of the keywords from the query it is considered to be a valid subgraph. Finally, we terminate when we have obtained top-k results that have a weight less than other valid subgraphs or any other path that can be expanded to become a valid subgraph.

### 1.3 Overview of the Thesis

In Chapter 2, we review related work. In Chapter 3, we describe our approach for keyword search over weighted keyword-augmented RDF graphs. Chapter 4 contains descriptions and results of the conducted experiments. The conclusion and possible future work are presented in Chapter 5.

# Chapter 2

## Literature Review

Many attempts have been made for searching RDF knowledge bases using keywords queries. Most of this work can be categorized in two main categories. The first category of works aims at mapping the keyword query into one or more structured queries. For instance, the authors in [12] assume that the user keyword-query is an implicit representation of a structured triple-pattern query. They try to infer such structured query using the RDF graph and retrieve the top-k most relevant structured queries. Then they provide the user with the retrieved queries and let her choose the most appropriate structured query to be evaluated. Their approach involves user interaction, and in addition suffers from a loss-of-information phenomenon since typically  $k$  is set to a small number. One way to overcome the problem of engaging the user in the inference process is to directly evaluate the top-k inferred queries. Again, this has the problem of information loss and in addition is typically inefficient since each one of these queries would have to be evaluated against the RDF graph.

The work on query inference from a user's natural language question in [13] is also closely related to the previous work. It utilizes natural-language processing tools and tries to parse a user's question in order to infer the most-likely structured query. Their technique however relies heavily on the quality of the parsing process and it also suffers from the information-loss problem highlighted above.

In the paper Natural Language Questions for the Web of Data [14], the authors again work on translating a natural language into a SPARQL query. The authors have seven steps that they follow in order to solve ambiguity and the issue of segmentation of queries. The authors convert and map questions into meaningful phrases, phrases to semantic entities, classes, and relations, and then construct SPARQL triple patterns. The authors make use of types, surface names and textual patterns provided by the knowledge base. The types help the disambiguation process, by which they are able to understand the category of the entity.

The second category of work tries to overcome the problem of information loss and efficiency issues of the query inference, by directly retrieving results of

the keyword query. The work on keyword search over XML data for instance, falls into this category. XKSearch [15] returns a set of nodes that contain the query keywords either in their labels or in the labels of their descendant nodes and have no descendant node that also contains all keywords.

Similarly, XRank [16] returns the set of elements that contain at least one occurrence of all of the query keywords, after excluding the occurrences of the keywords in sub-elements that already contain all of the query keywords.

However, the above mentioned techniques assume a tree-structure and thus cannot be directly applied to graph-structured data such as RDF graphs.

Also, closely related to our work is the language-modeling (LM) approach for keyword search over XML data proposed in [17]. The authors assume that a keyword query has an implicit mapping of each keyword into XML element(s). Their ranking is based on the hierarchical language-models proposed in [18] and they utilize the distribution of terms in the elements of the XML collection to give weights the different component of the LMs. However, the setting of XML data is quite different from that of RDF since in XML the retrieval unit is an XML document (or a subtree). In an RDF setting, we are interested in ranking subgraphs that match the user’s query. These subgraphs are not known in advance and are computed on the fly during retrieval time, and thus most of the prior work on XML IR would not apply.

Keyword search on graphs which returns a ranked list of Steiner trees [19, 20, 21, 22] (the exception is [23] which returns graphs) deals with the latter problem of having a predefined retrieval unit. However, the result ranking in each of the above is based on the structure of the results [19, 24] (usually based on aggregating the number or weights of nodes and edges), or on a combination of these properties with content-based measures such as tf-idf [25, 21, 23] or language models [7].

The BANKS system [19] enables keyword search on graph databases. Given a keyword query, an answer is a subgraph connecting some set of nodes that ”cover” the keywords (i.e., match the query keywords). The relevance of an answer is determined based on a combination of edge weights and node weights in the answer graph. The importance of an edge depends upon the type of the edge, i.e., its relationship. Node weights on the other hand represent the static authority or importance of nodes and are set as a function of the in-degree of the node. However, BANKS completely ignores the content during ranking, and thus does not make use of the content of the triples as an additional evidence of relevance to the query.

A closely related work that combines structure and content for ranking is the LM-based ranking model in [7] for ranking objects (entities in an RDF setting). The model assumes that each entity is associated with a set of records extracted from web sources. In turn, each record is associated with a “document”. The relevance of each “document” (and correspondingly, the entity associated with it) to a keyword query is estimated using LMs. This model however assumes that

the retrieval unit is entities only. While our ranking model goes beyond this to treat triples in a holistic manner by taking into account the relationships between the entities. In addition, it assumes the presence of a document associated with each Web Object or entity, something that we lack in the case of RDF data in general.

The Semantic Search Challenge provided a benchmark for keyword queries over RDF data, however the judgments were made over entities built by assembling all the triples that shared the same subject. The best performing approach [26] ranked the entities using a combination of BM25F and additional hand-crafted information about some predicates, properties and sites. In contrast, we retrieve the set of subgraphs that match the query keywords and rank them. We believe that the graph representation provides more concise answers to the user information need than a set of entities.

Keyword search on RDF graphs has also been addressed in [27]. While the authors provide a retrieval and a ranking model for keyword queries over RDF graphs, their algorithms are very graph dependent, meaning that they need a lot of indexing effort for each RDF graph it is run on. Moreover, their ranking model did not take into consideration the importance of the triples particularly with respect to the query keywords and focused mainly on the structure of the results. Finally, their model was computationally very expensive as it did not include a top-k processing approach but instead used the brute-force retrieve-then-rank approach.

Finally, the closest work to ours is [28], where the authors propose a graph summarization algorithm to summarize the large RDF data sets using the types of each node. They also provide an enhanced retrieval algorithm to search and retrieve results more efficiently from the summarized graph. On the other hand the authors do not provide a ranking model to retrieve the most relevant answer to the query and do not take into consideration predicate keywords in their retrieval algorithm.



# Chapter 3

## Our Approach

### 3.1 System Overview

Our knowledge base consists of a set of SPO-triples such as the one shown in Table 1.1. To be able to process keyword queries, we use the labels provided by the knowledge base. These labels provide for each URI from our knowledge graph a set of surface names. We create an inverted index of URIs and their surface names using Lucene. For example, the complete URI for Woody Allen would have "woody allen" as a surface name. We also retrieve a set of textual-patterns for each predicate (relation label) from our knowledge base by crowdsourcing. We then create an inverted index of the textual-patterns and their predicates.

To be able to rank the sub-graphs, we associate each triple with a weight reflecting the importance of the triple. We also refer to these weights as witness-count of a triple. These weights can be computed in various ways depending on the nature of the underlying knowledge base. For instance, for Wikipedia based RDF graphs such as DBPedia and Yago, we can utilize the link structure of the corpus from which the RDF graph was constructed to compute the weights for the triples. That is, the weight of triple  $t = (s, p, o)$  can be set to the number of Wikipedia articles that link to both  $s$  and  $o$ . This way, the weight of a triple reflects the number of Wikipedia pages that mention both the subject and the object of the triple. In other words, it estimates the number of pages that convey the fact represented by the triple. The higher this weight is, the more important or popular the fact is. We do this for every triple in the knowledge base and we coin the resulting graph *weighted keyword-augmented RDF graph*.

To retrieve a set of sub-graphs given a keyword query, first, we process the query; identify the predicates and separate them from the entities with a segmentation and resource matching approach explained in Section 2. Then, using the Backward Search algorithm explained in Section 3, we retrieve all sub-graphs that match the given query. Finally, Section 4 contains detailed explanation on how the scoring function works and how we give weights for each and every single

SPO triple. Section 5 contains explanation on how the algorithm terminates and how we confirm that we extracted the top-k most relevant sub-graphs.

## 3.2 Query Segmentation and Resource Matching

In this section we describe in details the steps we perform for query segmentation and resource matching. We explain how we separate the entities and relations (predicates) from each other. We also describe how we match and select the most relevant URI for each given phrase (an entity or a predicate).

---

### Algorithm 1 Query Segmentation - Resource Matching

---

**Input:**  $Q = \{w_1, w_2, \dots, w_m\}$ ,  $S$ ,  $P$ ,  $G = \{V, E\}$   
**Output:** sets  $q$  and  $r$   
Initialize  $tq \leftarrow \emptyset, q \leftarrow \emptyset, r \leftarrow \emptyset, a \leftarrow \emptyset, W = null$   
**for**  $i = 1..m$  **do**  
  **if**  $w_i \in P$  **then**  
     $tr \leftarrow P(w_i)$   
     $a_i \leftarrow (w_{i-1}, P(w_i), w_{i+1})$   
    **if**  $W \neq null$  **then**  
       $tq \leftarrow W$   
    **end if**  
  **else**  
     $W \leftarrow w_i$   
  **end if**  
**end for**  
 $W \leftarrow null$   
 $q \leftarrow EntityMatching(W, tq, S)$   
**for**  $j = 1..size(tr)$  **do**  
  **if**  $size(tr_j) > 1$  **then**  
     $\forall u \in tr_j \exists v$  where  $degree(w_{j-1}, v, w_{j+1}) > degree(w_{j-1}, u, w_{j+1})$   
     $r \leftarrow v$   
  **else**  
     $r \leftarrow tr_j$   
  **end if**  
**end for**

---

Algorithm 1 contains the steps that we follow in order to successfully separate entities from relations. The algorithm takes as an input the set of phrases of the query  $Q$ , and outputs two sets: one containing a set of URIs for the surface-names, while the other a set of URIs for the textual-patterns. The data structures used in the algorithm are two empty sets  $tq$  and  $tr$  used as auxiliary sets for temporary

---

**Algorithm 2** EntityMatching

---

**Input:**  $tq, W, S$

**Output:** set  $q$

**for**  $i = 1..size(tq)$  **do**

$W \leftarrow tq_i$

**if**  $W \in S$  **then**

**if**  $size(S(W)) > 1$  **then**

$\forall u \in S(W) \exists v$  where  $degree(v) > degree(u)$

$q \leftarrow v$

**else**

$q \leftarrow S(tq_i)$

**end if**

**else**

**while**  $W \neq \text{null}$  **do**

            split  $W$  starting from the most right

**if**  $W \in S$  **then**

**if**  $size(S(W)) > 1$  **then**

$\forall u \in S(W) \exists v$  where  $degree(v) > degree(u)$

$q \leftarrow v$

**else**

$q \leftarrow S(tq_i)$

**end if**

**end if**

**end while**

**end if**

**end for**

---

use.  $a$  is used to track the entities with their relations, while  $W$  to create the surface-names needed to link to the URI.  $P$  maps each textual pattern to a predicate, while  $S$  maps a surface name to a URI.

In the first loop of the algorithm we start by marking each query phrase as a predicate or a non-predicate - by trying to match each word to a textual pattern. Then, using "EntityMatching" function we try to match the non-predicates to labels by considering all the words in between two predicates to be a complete phrase - if we had no match for the phrase we split the phrase into different parts in different ways. We do that using a greedy approach starting from the most left and removing the last word added to the complete phrase. If we end up getting more than one match for a surface-name we consider the one with the highest degree (number of times the label appears in a triple in our graph).

In the final loop of the algorithm, and after matching all the surface names to their URIs we check if there exist textual patterns that match for more than one predicate. We calculate the frequency of predicates occurring in the knowledge base considering that the entities next to the predicate in the query are either the subject or the object of the given predicate; we pick the one with the highest degree. The entities next to a predicate are marked in the first loop and later matched to their URIs in the second loop. If neither the right nor the left side of the predicate are entities we consider the entities next to the predicates neighboring predicates. Consider the example; given a query

$Q = \text{"woody allen judy davis actor"}$

The query  $Q$  contains a predicate, and two different labels (surface names) each composed of two words.

We start by marking the phrases as entities or predicates. The first word in this query is *woody*, this word does not exist in the set of textual patterns, hence we add the word in  $W$  with an underscore behind it, and proceed to the next word. The next word is *allen*, which again does not match to a textual pattern, we concatenate it with  $W$  and add an underscore at the end of the newly concatenated phrase ( $W = \text{woody\_allen\_}$ ). The next three words are also added to  $W$  separated with underscores since they are not textual patterns.

Now, the next word is *actor*, which exists in the set of textual patterns, we add its corresponding predicates (in this case only one) *actedIn* to the temporary set  $tr$ , and add  $W$  to the set of temporary labels  $tq$  (after removing the last underscore), and then we empty  $W$ .

At this stage we have:

- $tr = \{ \text{actedIn} \}$
- $tq = \{ \text{woody\_allen\_judy\_davis} \}$

Now, we continue with the next step (Algorithm 2): mapping surface names

to their corresponding URIs. We only have one phrase in the set, we try to match it to a URI, but since we do not have a match for the complete phrase we remove the last word (the one after the last underscore) and we try to match the phrase again. For our example *woody\_allen\_judy* would be the next try, followed with *woody\_allen* which is a hit for a matching URI i.e.,  $\langle \text{Woody\_Allen} \rangle$  - in the case of YAGO - and we add it to the set  $q$ . We remove the matching part of the phrase and continue with the remaining part *judy\_davis* where in our case this is a hit with a single URI and we add it to the set  $q$ .

At this stage we now have:

- $tr = \{ \{ \text{actedIn} \} \}$
- $q = \{ \langle \text{Woody\_Allen} \rangle, \langle \text{Judy\_Davis} \rangle \}$

We proceed with the predicates, and in this example we had one hit for the textual pattern. Hence, we add *actedIn* to the set  $r$ . If we had more than one hit to a predicate we calculate the number of times the predicates occurred in the knowledge base considering that the entities next to the predicate in the query are either the subject or the object of the given predicate. Here we would count how many times  $\langle \text{Judy\_Davis} \rangle$  has appeared in the knowledge base as a subject or an object having *actedIn* as a predicate or the other predicate and pick the predicate having the higher result.

### 3.3 Retrieval Model

Our retrieval model is based on the backward search heuristic. The backward search starts from each vertex in the graph  $G$  that matches a keyword from the query, it then expands the neighbors of each vertex until a relevant answer is found and the termination condition is met.

From the related work, Scalable Keyword Search On Large RDF Data [28] the authors have applied the backward search idea. Their method does not take into consideration the relation (edges/predicates) in between the entities. This prohibits a user from getting results when searching for queries containing only predicates, e.g., "actor directors". Also, they do not have a scoring function to retrieve the most popular and relevant results at top.

In our model we separate the entities from relations and then start exploring the graph starting from the entities and expanding through the relationships found in the query; Meaning we extract only results containing one of the relations found in the query (if there exists one). If there are no entities in the query and only relations as the example given above, we expand the neighbors of entities having the highest witnesscount with one of the relations as an edge. Algorithm 3 shows how we retrieve the top-k most relevant result.

Given  $q = \{q_1, q_2, \dots, q_m\}$  (the set of entities),  $r = \{r_1, r_2, \dots, r_m\}$  (the set of the predicates), and  $G = V, E$  (our RDF graph). We initialize  $m$  empty priority

---

**Algorithm 3** Backward Search

---

**Input:**  $q = \{q_1, q_2, \dots, q_m\}$ ,  $r = \{r_1, r_2, \dots, r_m\}$ ,  $G = \{V, E\}$

**Output:** top- $k$  retrieved sub-graphs

$a$  min-heaps  $\{a_1, \dots, a_m\}$ ;

$M \leftarrow \emptyset$

**for**  $i = 1..m$  **do**

$v = q_i$

**for**  $\forall u \in V$  and  $d(v, u) \leq 1$  **do**

$a \leftarrow (v, p \leftarrow \{v, E(v, u), u\}, w(p) \leftarrow w(v, u))$

**if**  $u \notin M$  **then**

$M[u] \leftarrow \{\text{nil}, \dots, ((v, E(v, u)), w(u, v)), \dots, \text{nil}\}$

**else**

$M[u] \leftarrow ((v, E(v, u)), w(u, v))$

**end if**

**end for**

**end for**

**while** *termination condition not met* **do**

$(v, p, w(p)) \leftarrow \text{pop}(\arg \min_{i=1}^m \{\text{top}(a_i)\})$

**for**  $\forall u \in V$  and  $d(v, u) = 1$  and  $u \notin p$  **do**

$a \leftarrow (u, p \cup \{(E(v, u), u)\}, w(p) \leftarrow w(p) + w(u, v))$

**if**  $u \notin M$  **then**

$M[u] \leftarrow \{\text{nil}, \dots, ((v, E(v, u)), w(p)), \dots, \text{nil}\}$

**else**

$M[u] \leftarrow ((v, E(v, u)), w(p))$

**end if**

**end for**

**end while**

---

queues (e.g., min-heaps)  $\{a_1, \dots, a_m\}$ , one for each URI.  $a$  has the size of  $q$  multiplied by the size of  $r$  where each node of  $a$  contains all the expanded values of the vertex at that specific index with a specific relation (if  $r$  is not empty). Consider the example "Woody Allen's wife's birthday", here  $q = \{\text{"woody allen"}\}$ , while  $r = \{\text{"isBornOn"}, \text{"isMarriedTo"}\}$ . We only have 1 entity in  $q$  but 2 predicates in  $r$ , hence  $a$  will contain 2 min-heap in which all the expanded neighbors will be stored in increasing order of their scores. The first heap will be used to score "woody allen ismarriedto" while the second "woody allen isBornOn". Other than keeping the path and the newly expanded neighbor in  $a$  we also keep track of the predicates for each entity.

We also maintain a set  $M$  of elements, one for each distinct node we have expanded or explored so far in the backward search, to track the state of the node with their best known weights.  $M[u]$  is also used to identify complete and incomplete sub-graphs. The pair  $(vertex, relation), weight$  in the  $j^{th}$  entry of  $M[u]$  indicates a path from  $u$  to the vertex with its total weight; we also consider that there could be multiple entries in the same index, hence we add each entry of  $M$  is an array.

Consider the following example  $M[u'] = \{((v_5, p_5), 0.5), ((v_1, p_1), 0.6), nil\}$  in  $M$ . The entry indicates that  $v_5$  has been reached from  $q_1$  through  $p_5$ , while  $v_1$  from  $q_2$  through  $p_1$ . However, the  $3^{rd}$  position has not been reached by any expansion from any vertex containing  $q_3$  yet, hence  $M[u']$  is not a complete sub-graph yet.

The query entered by the user is composed of the entities and predicates. The number of entities and the predicates affect the performance of the algorithm and the relevance of the resulting sub-graphs. Below are the five different types of queries that a user might enter.

- Multiple entities and no predicates.
- Multiple entities with predicates.
- One entity without predicates.
- One entity with predicates.
- Multiple predicates and no entities.

For multiple entities and no predicates, the neighbors of the entities are expanded with no condition on the relationship with the parent node. The edges of the graph are not used. Similarly for a query containing one entity and no predicates, the relation of the nodes are not used. The difference with the previous case is that the algorithm terminates when all the values remaining to be expanded contain a path with more than 2 nodes or when the termination condition applies.

For multiple entities with predicates, again the expansion process is similar to the one above but we only consider sub-graphs to be candidate answers (complete)

when they contain all the entities and all the predicates in their path. Similarly for a query containing one entity and predicates, a sub-graph is considered to be a candidate answer only if it contains all the URIs of the surface-names and textual-patterns. Again the difference with the previous case is that it terminates when all the values remaining to be expanded contain a path with more than 2 nodes or when the termination condition applies.

For multiple predicates and no entities, we initialize  $W$  with a limited number of entities - the ones with a higher witnesscount - that have an edge with one of the predicates from the query. The last entity with the least witnesscount will then be used to check if the given number of entities guarantee the best results. If not, a new set of entities will be passed to the algorithm until all nodes are passed or the termination condition applies.

The algorithm takes as input the complete set of the URIs of the surface-names as well as the complete set of the URIs of the textual-patterns. The algorithm proceeds in iterations. In the first iteration, for each vertex  $v$  from  $q_i$  to  $q_m$  and every neighbor  $u$  of  $v$  (including  $v$  itself), we add an entry  $(v, p, u, w(u, v))$  to the priority queue  $a_i$  (entries are sorted in the ascending order of  $w(p)$  where  $p$  stands for a path and  $w(u, v)$  represents the weight of the triple containing  $u$  and  $v$  as Subject or Object from the knowledge base). We also add  $(v, w(u, v))$  to  $M[u][i]$ .

The second iteration of our algorithm starts by popping the smallest top entry  $(u, p = v, \dots, u, w(p))$  from the queue  $a_i$  of  $\{a_1..a_m\}$ . For each neighbor  $u'$  of  $u$  in  $G$  such that  $u'$  is not in  $p$ , we push an entry  $(u'; p \cup \{u'\}, w(p) + w(u, v))$  to the queue  $a_i$ . We also update  $M$  with  $u'$  meaning  $M[u'][i] = (v, w(p) + w(u, v))$ .

If an entry  $M[u]$  for an entity  $u$  has no nil values, then this entry with all its pairs contains a sub-graph that is a candidate answer (complete sub-graph).

In the worst case the whole graph  $G = (V, E)$  is explored in finding the top-k most relevant answers, where our method has to traverse every triple in  $G$  for each of the matched keywords  $q = \{w_1, w_2, \dots, w_m\}$  leading to an overall cost of  $O(m \cdot E)$ .

Consider an example, given a user query "woody allen judy davis actors" after the stage of segmentation and resource matching we would get the two sets  $r$  and  $q$  as inputs to the algorithm. The set  $q$  would contain the two URIs  $\{ \langle \text{Woody\_Allen} \rangle, \langle \text{judy\_davis} \rangle \}$ , while the set  $r$  would contain the URI  $\{ \text{"actedIn"} \}$ . The first loop of the algorithm will iterate  $n$  times, where  $n$  is the size of the set  $q$  (number of entities). All the direct neighbors of the entities in  $q$  will be expanded in the first loop. In this case entities such as "Vicky Cristina Barcelona" will be expanded. The expanded nodes will be added to  $M$  and the min-heap  $a$  each with their scores (after calculating the scores; explained in Section 3.4). In this case.  $M[\text{Vicky Cristina Barcelona}] = \{ (\text{WoodyAllen}, \text{actedIn}, \text{Score}) \}$ , and we insert in  $a$ ,  $(\text{WoodyAllen}, (\text{Woody Allen}, \text{actedIn}, \text{Vicky Cristina Barcelona}), \text{Score})$ . This entry of  $M$  is not a candidate since it does not contain all the predicates from  $r$  in its path. As you notice we keep



track of the predicates for each path. After expanding all the direct neighbors of the vertex we continue to the second loop. In the second loop we remove from  $a$  the element containing the minimum score. We expand the neighbors of the direct neighbor of our entity and we store it in  $M$  and  $a$  but by adding the score of the newly expanded neighbor to the score that is already in  $a$ . In this case one of the expanded neighbors will be a literal that is the age of Louise Lasser. The algorithm terminates when it guarantees that the scores of our top-k sub-graphs are the lowest that can ever be created.

Consider the example "Germany England" as a query. The retrieval model without a scoring function (explained in Section 3.4) will produce a top-6 result as some in Table 3.1.

Subject (S)	Predicate (P)	Object (O)
Germany	type	country
England	type	country
Germany	type	economy
England	type	economy
Germany	participatedIn	Battle of Graveney Marsh
Battle of Graveney Marsh	happenedIn	England
Germany	participatedIn	Adlertag
Adlertag	happenedIn	England
William Cramer	isCitizenOf	Germany
William Cramer	livesIn	England
Peter Griess	isCitizenOf	Germany
Peter Griess	livesIn	England

Table 3.1: A Subgraph for the query "Germany England"

### 3.4 Scoring Function

To address the ambiguities of entities and enable ranking, we associate each edge in the graph with a weight reflecting the importance of the edge - we call this weight witnesscount, i.e., weighted keyword-augmented RDF graphs. The total weight of the path when a neighbor is expanded (parent-neighbor) can be computed as:

$$w(t) = \alpha * \left(1 - \frac{wc(t)}{\sum_{t' \in KB} wc(t')}\right) + (1 - \alpha) \frac{degree(s) + degree(o)}{\sum_{t' \in KB} degree(s') + degree(o')}$$

where  $\alpha$  is a weighting parameter between 0 and 1,  $wc(t)$  is the witness count of the triple  $t$  (parent and expanded neighbor),  $KB$  represents the whole knowledge base,  $degree(e)$  is the number of triples with  $e$  as subject or object, and  $t' = (s', r', o')$ . Finally, given a path  $p$  with distance  $d(p)$  and assuming the case of

adding  $u$  to  $p$  by expanding through triple  $t$ , we update the distance of  $p$  as follows:

$$d(p) = d(p) + w(t)$$

The first part of the equation takes into consideration the popularity of the triple which is its importance, since we are using the witnesscount of the triple that is the number of links pointing to both the entities in the triple. This part helps in retrieving sub-graphs that contain in them entities that are the most popular. For example, a sub-graph containing Albert Einstein would have a higher rank than that to Max Steenbeck, since the witnesscount of Albert Einstein would be much higher compared to Max Steenbeck and hence will have a higher score.

The second part of the equation considers the degree of the triple (the number of times the parent (Subject) and the expanded neighbor (Object) occur in the knowledge base). For Example, Albert Einstein would have facts such as:

”Albert Einstein hasGender male”

”Albert Einstein isMarriedTo Elsa Einstein”.

Both facts are relevant and are candidate sub-graphs to our query. We want to get triples such as the one containing information on who Albert Einstein is married to on top and results such as the gender to be at the bottom. Hence we give triples with entities occurring a lot in the knowledge base a higher score.

### 3.5 Termination Condition

The top-k results retrieved using our approach, should provide us with the most relevant results of the query. We need to insure that it is not possible to get a new relevant sub-graph having a score less than the score of the  $k^{th}$  candidate. In order to guarantee the above mentioned condition, we should consider two cases. First, when an existing incomplete sub-graph - a non-candidate - is transformed to a candidate, it should have a weight greater than that of all the top-k candidates. Second, a newly created sub-graph should not have a weight less than that of the top-k results.

We call nodes that are in  $a$  but not in  $M$  unseen nodes. We consider two cases before terminating the algorithm:

- Unseen nodes form a new candidate sub-graph.
- A non-candidate sub-graph is completed with a node and a candidate sub-graph is formed.

We make sure that are top-k results are the k most relevant results by calculating  $\tau$  and  $\min(\phi)$  and comparing them with our  $min_k$ . When our  $min_k$  is less than the other two we terminate. In the spirit of a top-k approach, we terminate when the following condition holds:

- Let  $min_k$  = the  $k^{th}$  smallest score of a candidate answer, i.e., those in  $M$  with no nil values.
- Let  $\tau = \sum_{i=1}^m w(p_i)$ , where  $w(p_i)$  is the weight of the head of priority heap  $a_i$
- Let  $\phi_v = \sum_{i=1}^m w_i * b_i + w(p_i) * (1 - b_i)$  where  $b_i$  is equal to 0 if  $M[v][i] = nil$  and 1 otherwise,  $w_i$  is the weight of the path at the  $i^{th}$  entry of  $M$ ,  $w(p_i)$  is the weight of the head of priority heap  $a_i$ . This should be computed only for those vertices in  $M$  that are not fully explored (i.e., whose entries in  $M$  contain nil values - non-candidates).
- let  $\phi = min(phi_v)$  over all the nodes where  $phi_v$  is computed.
- Terminate whenever  $min_k$  is less than or equal to  $min(\tau, \phi)$ .

Consider the following example:

let  $m = 2$  and  $k = 2$

$A_1 = (v_1, 0.55), (v_2, 0.77), (v_3, 0.85)$

$A_2 = (v_4, 0.2), (v_5, 0.7), (v_9, 0.9)$

$\tau = 0.55 + 0.2 = 0.75$

candidates:

$$M["Y"] = \{(v_2, 0.77), (v_5, 0.7)\}$$

$$M["Z"] = \{(v_1, 0.55), (v_9, 0.9)\}$$

non-candidates:

$$M["X"] = \{(v_2, 0.77), nil\}$$

$$M["W"] = \{nil, (v_4, 0.2)\}$$

$Weight_z = 0.77 + 0.7 = 1.45$

$Weight_y = 0.55 + 0.9 = 1.42$

$Weight_w = top(a_2) + 0.2 = 0.55 + 0.2 = 0.77$

$Weight_x = 0.77 + top(a_2) = 0.77 + 0.2 = 0.97$

$min_k = 1.45$

$\phi = 0.77$

terminate only when  $min_k < min(\phi, \tau)$

# Chapter 4

## Experiments

Our experiment is composed of several stages. We test our algorithms and models on a very large real-world RDF graph, called YAGO. We compare the top-10 results of our approach with different alpha values for the scoring function. After which we compare the top-10 results of three different approaches that search RDF graphs using keywords with our method.

First, we implemented the three different approaches and methods for searching RDF graphs using keywords only. Then, we run the algorithms on a set of keyword queries and place the top-10 results of each approach in a pool later to be evaluated. After which we ask some volunteers to evaluate the results of each query. Finally, we calculate the NDCG value for each method.

This Chapter is organized as follows. Section 4.1 contains details about what dataset we used. Section 4.2 contains descriptions about three other methods used for experimentation. Section 4.3 shows how we performed our experiments. Section 4.4 contains results of the experiments performed.

### 4.1 Dataset

In this thesis we use the resources of YAGO dataset [3] to do our testing and evaluations. YAGO is a large-scale general-purpose RDF dataset derived from Wikipedia and WordNet. It contains more than 120 million triples about 10 million resources (like persons, organizations, cities, etc.).

We made use of the labels provided by YAGO that contain surface names for each entity in the dataset. We created an inverted index for each surface name and its entity.

The dataset is composed of 75 distinct predicates, for each predicate we found a set of textual-patterns by crowdsourcing.

We use Apache Lucene to create an inverted index for the resources of YAGO, the surface-names, and the textual-patterns.

## 4.2 Competitions

In the below subsections we explain the three approaches used to compare with our proposed approach. The three approaches are: Scalable Keyword Search on Large RDF data [28], Web Object Retrieval [7], and Keyword Search Over RDF [27].

In order to be able to rank the sub-graphs for the two approaches "Web Object Retrieval" and "Keyword Search Over RDF", we use the same *weighted RDF graph* and further augment each triple with a new set of keywords. These keywords are the links present in the pages of both the entities in a triple, combined with the textual patterns of the predicate *weighted keyword-augmented RDF graph*. We also add weights for each keyword of each triple - the weights are the number of links pointing to that keyword.

### 4.2.1 Scalable Keyword Search on Large RDF data

In the paper Scalable Keyword Search On Large RDF Data [28] the authors have applied the backward search idea. Their method does not take into consideration the relation in between the entities (the predicates) and does not use any scoring function to rank the sub-graph. For example, if we run the query "scientists graduated from Boston University" we get the 5 results found in Table 4.1 as top-5 results. The top results shows in Table 4.1 are not scientists who graduated from Boston University but are scientists who worked at the Boston University.

<b>Subject (S)</b>	<b>Predicate (P)</b>	<b>Object (O)</b>
Anna Geifman	type	Scientist
Anna Geifman	worksAt	Boston University
Peter L. Berger	type	Scientist
Peter L. Berger	worksAt	Boston University
H. Eugene Stanley	type	Scientist
H. Eugene Stanley	worksAt	Boston University
Carol Christian	type	Scientist
Carol Christian	graduatedFrom	Boston University
Julius Sumner Miller	type	Scientist
Julius Sumner Miller	graduatedFrom	Boston University

Table 4.1: Backward Search results

For this approach we use the same weighted keyword-augmented RDF graph.

### 4.2.2 Web Object Retrieval

The method Web Object Retrieval is based on entities, meaning the top-k results produced by this algorithm will be entities rather than sub-graphs. In this method

we start by splitting the user query into words and then we go through word-by-word and extract all the triples containing that keyword. Then, we calculate a score for each triple, followed with a score for each entity in each triple. We calculate the score using the following formula

$$Score(e) = P(q_i|t) * P(t|e)$$

$$P(q_i|t) = \frac{wc(t, q_i)}{\sum_{t' \in KB} wc(t', q_i)}$$

where  $wc(t', q_i)$  stands for the weight (witnesscount) of the keyword found in the triple.

Finally we output the top-k entities as relevant results to our keyword query. Table 4.2 shows the top-5 results found after running the query: "scientists graduated from Boston University". Most of the results are non relevant since in this approach each keyword is treated separately, which causes a lot of ambiguity.

1	New York City
2	Harvard University
3	1998-12-09
4	Julius Sumner Miller
5	David Winning

Table 4.2: Entity Based Results

### 4.2.3 Keyword Search Over RDF

The third method is based on the Keyword Search Over RDF [27]. This method returns the top-k sub-graphs and not entities. The authors first retrieve triples that contain the keywords of the user query, and then proceed to forming sub-graph with one or more triples. Sub-graphs can be formed by triples if the following three rules apply:

- The triples have a similar entity (the subject or the object)
- The sub-graph formed must be unique and maximal, i.e., a sub-graph created must not be a subset of any other sub-graph.
- Sub-graphs should contain triples matching different sets of keywords, i.e., triples in the same sub-graph must not have the same set of keywords or a subset of keywords of a triple in the sub-graph. If they do they should be part of two different and separate sub-graphs.

For example consider the user query "scientists graduated from Boston University".

Subject (S)	Predicate (P)	Object (O)	Keywords
Fred Joseph	graduatedFrom	Harvard University	graduat: 13703, boston: 1113, univers: 28829
Fred Joseph	wasBornIn	Boston	scientist: 15, boston: 14789
John Andrew Sullivan John Andrew Sullivan	wasBornIn graduatedFrom	Boston Boston University	boston: 278, univers: 322 scientist: 15, boston: 15083, graduat: 1500
Fred Richmond Fred Richmond	graduatedFrom wasBornIn	Boston University Boston	boston: 3179, univers: 3482 scientist: 15, boston: 11422, graduat: 2870
H. Eugene Stanley H. Eugene Stanley	gradatedFrom worksAt	Wesleyan University Boston University	graduat: 9827, scientist: 240 boston: 2834, univers: 2442
Carol Christian	graduatedFrom	Boston University	boston: 1475, univers: 4546, graduat: 3487
Carol Christian	type	Scientist	scientist: 570

Table 4.3: Backward Search results

Table 4.3 contains the top-5 results for the above example. The valid sub-graphs are the ones that have a match for all the keywords from the user query. The first sub-graph matches all the keywords of the user query but is not a relevant answer to our query. From the top-5 the only relevant answer appears to be the 5<sup>th</sup> sub-graph.

### 4.3 Experimental Setup

After implementing three of the methods, we created a website for the evaluation phase. We then created a benchmark composed of 35 queries found in Appendix B. 3 of the 35 queries were gold queries found in Appendix C, meaning they were added intentionally in order to make sure the evaluator has done her job correctly. After having the benchmark ready, we ran all four algorithms. The same benchmark queries are used for all four methods. We also ran our new approach with different alpha values - for alpha 0 to 1 with a scale of 0.1. We then combined all the top 10 results and added them to our database later to be evaluated. Finally, we asked 5 people to evaluate our queries based on the given guidelines found in Appendix D, in between four different rates:

- highly relevant and popular
- highly relevant
- marginally relevant
- non relevant.

There were 1422 results for the above 35 queries, it took 3 days for the evaluators to finish.

## 4.4 Experimental Results

We extracted the evaluations of all the queries, and checked the evaluations of the gold queries. The participants evaluated all of the gold queries correctly. Then, we calculated the Fleiss' kappa [29] value for evaluating the reliability of agreement between our participants using the formula:

$$\kappa = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e}$$

We got  $\bar{P}_e = 0.326$ ,  $\bar{P} = 0.775$ ,  $\kappa = 0.66$  which is in the range of substantial agreement.

We used the majority vote from the 5 evaluators in order to calculate the NDCG [30] value for our backward search approach with different alpha values. We considered highly relevant and popular to have a relevance rate of 3, while highly relevant of 2, marginally relevant of 1 and finally non relevant 0. We then computed the normalized discounted cumulative gain using the formula:

$$NDCG = \frac{DCG}{IDCG}$$

$$DCG = rel_1 + \sum_{i=1}^n \frac{rel_i}{\log_2(i)}$$

where rel is the graded relevance of the result at position i

We use the discounted cumulative gain instead of using cumulative gain in order to penalize the highly relevant results appearing lower in a search result list as the graded relevance value is reduced logarithmically proportional to the position of the result.

IDCG stands for ideal discounted cumulative gain, which is the sorted version of the list of results based of their relevance.

We calculated the NDCG for all alpha from 0 to 1 in order to be able to choose the best value for alpha. The results of NDCG values for alpha from 0 to 1 are found in Figure 4.1.

From Figure 4.1 we notice that setting alpha to 0.3 gives the most relevant results as the top values. This proves the properties of the scoring function where when alpha is set to 0, the non popular results tend to go up. At a value of alpha = 0.3 the degree of the entities and the importance of the triple are normalized. Table 4.4 and 4.5 show the top-2 results for the query "woody allen scarlett johansson" setting alpha to 0.3 for Table 4.4, while 1 for Table 4.5. We notice



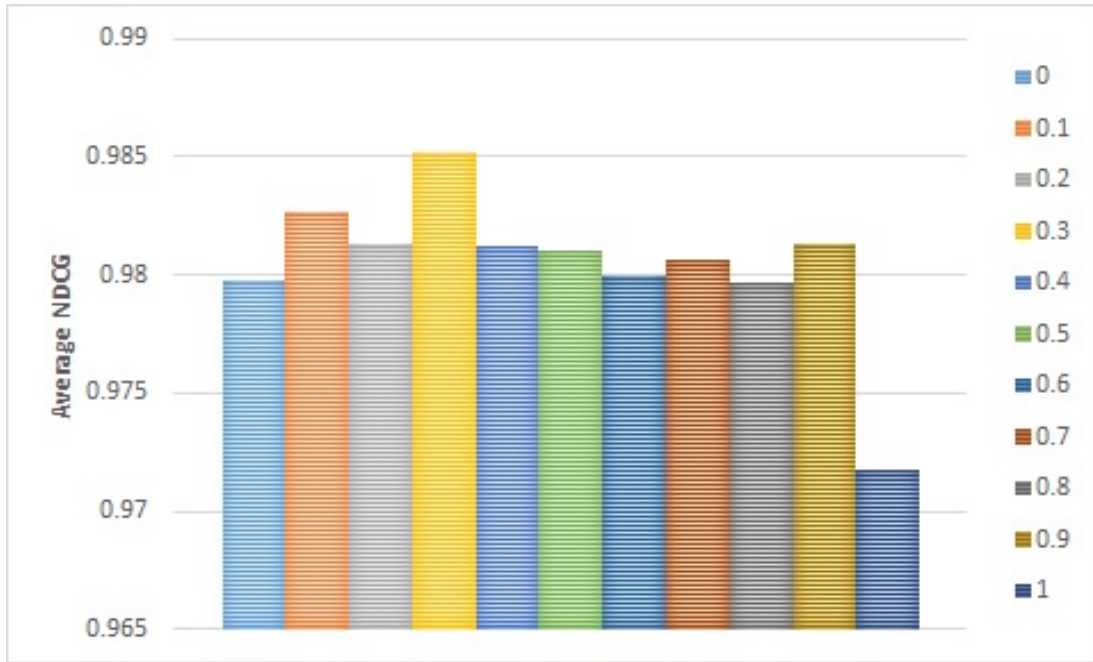


Figure 4.1: Average NDCG values with different alpha values

num	Subject (S)	Predicate (P)	Object (O)
1	Woody Allen Scarlett Johansson	directed actedIn	Vicky Cristina Barcelona Vicky Cristina Barcelona
2	Woody Allen Scarlett Johansson	created actedIn	Vicky Cristina Barcelona Vicky Cristina Barcelona

Table 4.4: alpha = 0.3, query = "woody allen scarlett johansson"

num	Subject (S)	Predicate (P)	Object (O)
1	Woody Allen Scarlett Johansson	directed actedIn	Vicky Cristina Barcelona Vicky Cristina Barcelona
2	Woody Allen Scarlett Johansson	type type	Jewish actors Jewish actors

Table 4.5: alpha = 1, query = "woody allen scarlett johansson"

from the results that a sub-graph containing the type is given more importance than a link between the two with a movie.

We now pick the evaluation results where alpha is set to 0.3 and compare it with the results of the other three methods, shown in Figure 4.2. Our method has gained the highest NDCG value. The "Web Object Retrieval" and "Keyword Search Over RDF" methods have a low NDCG, since they both produce a lot of ambiguous and non relevant results. The "Scalable Keyword Search on Large RDF data" method has a high NDCG value but not as high as our approach,

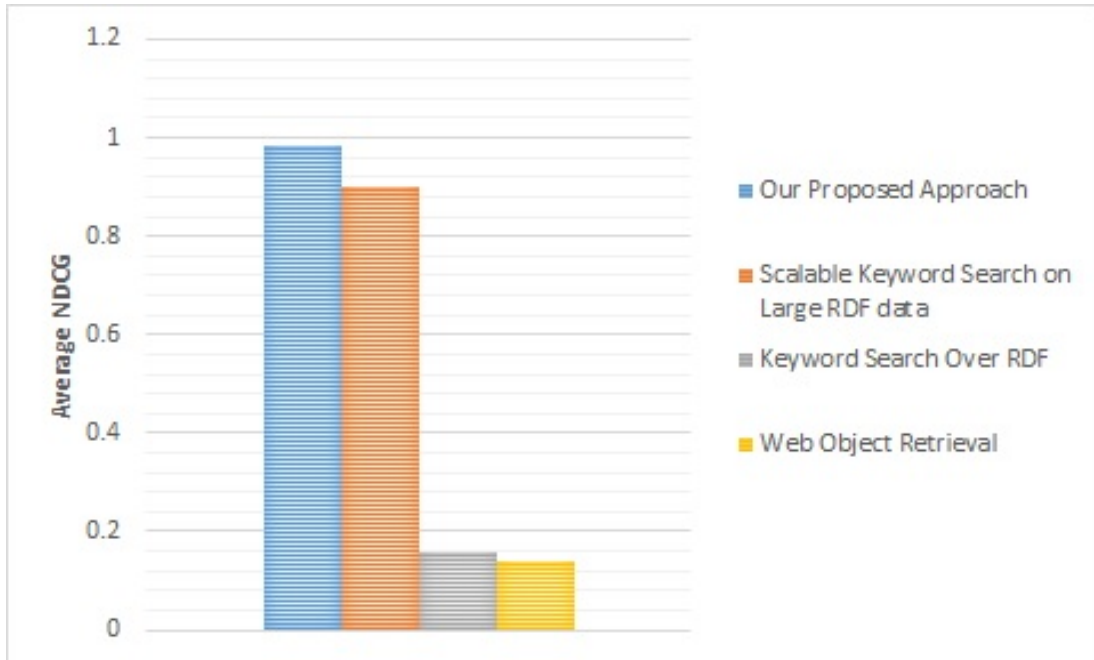


Figure 4.2: Average NDCG values for four approaches

since most of the top results produced by this method are relevant but are not ranked according to popularity, and relations are not taken into consideration.

Finally, we performed a t-test [31] which is used on two sets of data to determine if they are significantly different. We used the t-test to check if our approach is significantly different from the other three methods. The following are the p-values we got after performing the tests.

- Our approach with the backward search method. p-value = 0.026
- Our approach with the entity based method. p-value =  $2.52 \times 10^{-16}$
- Our approach with the Keyword Search on RDF data method. p-value =  $1.41 \times 10^{-15}$

The p-values of the last two methods are very small since the last two methods produce a lot of non-relevant answers. We conclude that our approach outperforms the others significantly.

# Chapter 5

## Conclusion and Future Work

In this thesis, we develop a retrieval model that enables users to search RDF graphs using *keywords only*. Our model takes as an input a keyword query and returns a ranked list of *RDF sub-graphs* most relevant to the given query. In order to do so, we performed the following tasks: First, we associated each entity in our knowledge graph with a set of keywords which are surface names (labels). Second, we addressed the ambiguities of entities and enabled ranking, by associating each edge in the graph with a weight reflecting the importance of the edge, i.e., weighted RDF graphs. Third we implemented an algorithm that would retrieve all the sub-graphs that match the keyword query from the underlying RDF graph.

We compared our model with four different methods to search large knowledge bases consisting of RDF graphs using keywords only. We evaluated the effectiveness of our retrieval model using a real-world RDF data-set and compared it to various state-of-the-art approaches for keyword search over RDF graphs. We have shown that our approach with a new ranking model produces the most relevant and popular results at the top. However, our retrieval model has an efficiency problem. The issue can be handled in many ways as discussed below.

Future work possibilities include the following:

- Improve the efficiency by reducing the number of neighbor entities extracted. This can be done by summarizing the RDF graph but in an effective way, in order not to have loss of important information. One possible way is to use the types embedded in the knowledge base, where each entity is associated with a type. The types can be used to understand which entities to expand in the summarized RDF graph.
- Predict best relationship in a query, for ambiguous keywords. This can be done again with the use of types where the types can be used to categorize the query.
- Run and test the algorithm on other datasets like DBpedia.

# Appendix A

## Abbreviations

KA	Keyword Augmented
KB	Knowledge Base
LM	Language model
RDF	Resource Description Framework
tf-idf	term frequencyinverse document frequency
URI	Uniform Resource Identifier

# Appendix B

## All Queries

- Ron Howard actor director
- Woody Allen actor director create
- Woody Allen's wife's birthday
- married Albert Einstein
- Woody Allen
- acted Judy Davis Woody Allen
- actor director Woody Allen Scarlett Johansson
- Albert Einstein Isaac Newton
- Boston University Albert Einstein
- Frank B. Morse Albert Einstein
- Judy Davis Woody Allen
- Taylor Swift Nicki Minaj Kendrick Lamar
- Woody Allen Scarlett Johansson
- actor director
- actors wife birthday
- born wife actor director
- Germany population
- What country has london as a capital

- elton john
- Dogma and Chasing Amy
- gone with the wind director
- Doctor Zhivago actors
- jessica alba Jennifer Aniston
- jessica alba Ioan Gruffudd
- Meryl streep mamma mia!
- Australia leader
- Russia leader wife birthplace
- England Capital population
- Germany England
- Pierce brosnan meryl streep
- leaders birthplace
- birthdays and date of death of actors

# Appendix C

## Gold Queries

- Lindsay Lohan birthplace population
- Tom Cruise Top Gun
- when did Salvador Dal die

# Appendix D

## Guidelines

You are given a set of natural language search queries where each query contains a set of results of two different forms.

- First Form

Example:

movies acted and directed by Woody Allen.

An answer:

Woody Allen directed To Rome with Love

Woody Allen actedIn To Rome with Love

Each result consists of one or more lines, where each line provides a fact related to the answer.

- Second Form

Example:

movies acted and directed by Woody Allen.

An answer:

To Rome with Love

Each result is an entity that should be a result for the query. If you are not familiar with the resulting entity or in case you are not sure whether the result is relevant to the query or not, you may click on it in order to be guided to its Wikipedia page.

Your task is to assess how the results are relevant to the query.

The relevance is projected on a three-level scale:

- Highly relevant and popular: You would select this level when the result is an answer you would expect for the given query. For example, for the query "movies acted by tom cruise" Both results "Tom Cruise actedIn Mission:



Impossible.” of the first form or ”Mission: Impossible” of the second form are highly relevant and popular.

- Highly relevant: You would select this level when the result is a correct answer but not a very popular one for the given query. For the same example as above, ”movies acted by tom cruise” Both results ”Tom Cruise actedIn Endless Love.” of the first form or ”Endless Love” of the second form are highly relevant.

Note: The difference in the above two is in the popularity of the result, as the example given above not all movies acted by Tom Cruise are that popular. The Highly relevant and popular would be for the most popular movies such as the film ”Mission: Impossible”, while movies like ”Endless Love” which are as well correct answers but not that popular would only be Highly relevant.

- Marginally relevant: You would select this level when the result is somehow related to the expected answer but does not fully answer the query. For example, for the query ”movies acted and directed by Woody Allen” ”Woody Allen directed Broadway Danny Rose. Nick Apollo Forte actedIn Broadway Danny Rose.” or ”The Floating Light Bulb” - which is a play written by Woody Allen - will be marginally relevant results to our example query.
- Non-relevant: You would select this level when the result is not suitable and is incorrect for the given query. That is, the statement does not provide a relevant answer to the given query. For example, for the query ”movies acted and directed by Woody Allen” both results ”Soon-Yi Previn isMarriedTo Woody Allen. SoonYi Previn wasBornOnDate 19701008.” or ”Elton John” for the second form will be non-relevant.

The order inside a line does not matter. For example: Soon-Yi Previn isMarriedTo Woody Allen and Woody Allen isMarriedTo Soon-Yi Previn are equivalent.

Note: If you did not understand the guidelines do not hesitate to contact me. Please do not randomly select answers, since it would appear in the Inter-rater agreement test and would complicate the project.

# Bibliography

- [1] A. Doan, L. Gravano, R. Ramakrishnan, and S. V. (Editors), “Special issue on managing information extraction,” *ACM SIGMOD Record*, vol. 37, no. 4, 2008.
- [2] S. Sarawagi, “Information extraction,” *Foundations and Trends in Databases*, vol. 2, no. 1, 2008.
- [3] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: A large ontology from wikipedia and wordnet,” *J. Web Sem.*, vol. 6, no. 3, 2008.
- [4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, “Dbpedia: A nucleus for a web of open data,” in *ISWC/ASWC*, 2007.
- [5] “Freebase.”
- [6] P. DeRose, X. Chai, B. J. Gao, W. Shen, A. Doan, P. Bohannon, and X. Zhu, “Building community wikipedias: A machine-human partnership approach,” in *ICDE*, 2008.
- [7] Z. Nie, Y. Ma, S. Shi, J. Wen, and W. Ma, “Web object retrieval,” in *WWW*, 2007.
- [8] “W3c: Resource description framework (rdf).” [www.w3.org/RDF/](http://www.w3.org/RDF/).
- [9] “Uniprot: Universal protein resource.” <http://www.uniprot.org/>.
- [10] J. Breslin, A. Passant, and S. Decker, *The Social Semantic Web*. springer, 2009.
- [11] C. Bizer, T. Heath, and T. Berners-Lee, “Linked data - the story so far,” 2010.
- [12] 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*, 2009.

- [13] V. Lopez, V. Uren, E. Motta, and M. Pasin, “Aqualog: An ontology-driven question answering system for organizational semantic intranets,” *Web Semant.*, vol. 5, no. 2, pp. 72–105, 2007.
- [14] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum, “Natural language questions for the web of data,” in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 379–390, Association for Computational Linguistics, 2012.
- [15] Y. Xu and Y. Papakonstantinou, “Efficient keyword search for smallest lcas in xml databases,” in *SIGMOD*, 2005.
- [16] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, “Xrank: ranked keyword search over xml documents,” in *SIGMOD*, 2003.
- [17] J. Kim, X. Xue, and W. B. Croft, “A probabilistic retrieval model for semistructured data,” in *ECIR*, 2009.
- [18] P. Ogilvie and J. Callan, “Hierarchical language models for xml component retrieval,” *Advances in XML Information Retrieval*, 2005.
- [19] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, “Keyword searching and browsing in databases using banks,” in *ICDE*, 2002.
- [20] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, “Bidirectional expansion for keyword search on graph databases,” in *VLDB*, 2005.
- [21] V. Hristidis, H. Hwang, and Y. Papakonstantinou, “Authority-based keyword search in databases,” *TODS*, vol. 33, no. 1, 2008.
- [22] K. Golenberg, B. Kimelfeld, and Y. Sagiv, “Keyword proximity search in complex data graphs,” in *SIGMOD*, 2008.
- [23] G. Li, B. Ooi, J. Feng, J. Wang, and L. Zhou, “Ease: an effective 3-in-1 keyword search method for unstructured, semistructured and structured data,” in *SIGMOD*, 2008.
- [24] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum, “Star: Steiner tree approximation in relationship-graphs,” in *ICDE*, 2009.
- [25] T. Cheng, X. Yan, and K. C.-C. Chang, “Entityrank: Searching entities directly and holistically,” in *VLDB*, 2007.
- [26] R. Blanco, P. Mika, and H. Zaragoza, “Entity search track submission by yahoo! research barcelona.” SemSearch, 2010.

- [27] S. Elbassuoni and R. Blanco, “Keyword search over rdf graphs,” in *CIKM*, CIKM '11, 2011.
- [28] W. Le, F. Li, A. Kementsietsidis, and S. Duan, “Scalable keyword search on large rdf data,” vol. 26, pp. 2774–2788, Nov 2014.
- [29] J. L. Fleiss, “Measuring nominal scale agreement among many raters.,” *Psychological bulletin*, vol. 76, no. 5, p. 378, 1971.
- [30] Y. Wang, L. Wang, Y. Li, D. He, W. Chen, and T.-Y. Liu, “A theoretical analysis of ndcg ranking measures,” in *Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013)*, 2013.
- [31] R. H. Browne, “The t-test p value and its relationship to the effect size and  $p(x_i, y)$ ,” *The American Statistician*, vol. 64, no. 1, pp. 30–33, 2010.