Keyword Search over RDF Graphs

Shady Elbassuoni *†
Max-Planck Institute for Informatics
Saarbrücken, Germany
elbass@mpii.de

Roi Blanco
Yahoo! Labs
Barcelona, Spain
roi@yahoo-inc.com

ABSTRACT

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 paper, we propose a retrieval model for keyword queries over RDF graphs. Our model retrieves a set of subgraphs that match the query keywords, and ranks them based on statistical language models. We show that our retrieval model outperforms the-state-of-the-art IR and DB models for keyword search over structured data using experiments over two real-world datasets.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*retrieval models, search process*

General Terms

Algorithms, Design, Experimentation

1. INTRODUCTION

The continuous growth of knowledge-sharing communities like Wikipedia and the advances in automated information-extraction from Web pages [5, 23] have made it possible to build large-scale knowledge bases. Examples of such knowledge bases include YAGO [26], DBpedia [1] and Freebase [7]. These repositories contain entities such as people, movies, books, etc. and the relationships between them such as `bornIn` , `actedIn` , `isAuthorOf` and so on. Such data is typically represented in the form of subject-predicate-object triples of the Semantic-Web data model RDF [22], 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 edges denoting predicates. Table 1 shows a set of RDF triples from a movie knowledge base.

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: A set of RDF triples from a movie knowledge base

RDF data can be queried using a conjunction of *triple patterns*, where a triple pattern is a triple with variables and the same variable in different patterns denotes a join condition. For example, the information need of finding comedies that have won the Academy Award can be expressed using the following 2 triple-patterns: `<?x hasGenre Comedy; ?x hasWonPrize Academy_Award>.`

Structured queries like the one above are very expressive, yet 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.

In this paper, we develop a retrieval model that enables users to search RDF graphs using *keywords*. Our model takes as an input a keyword query and returns a ranked list of *RDF subgraphs*. By retrieving subgraphs instead of just entities, we treat triples in a holistic manner and we explicitly take into account the relationships between the entities. This can be particularly beneficial for both result retrieval and result representation. As an example, consider running the query "comedy academy award" against the RDF collection in Table 1. One possible result to such a query retrieved by our model is the 2-triple subgraph `<Innerspace hasGenre Comedy; Innerspace hasWonPrize Academy_Award>.`

To be able to process keyword queries over RDF graphs, we associate each triple with a set of keywords derived from the subject and object of the triple, as well as representative keywords for the

*Work performed while intern at Yahoo! Research.

†This work is partially supported by the EU Large Scale Integrated Project LivingKnowledge (contract no. 231126).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...$10.00.

predicate. We explain how we do this in Section 3. To retrieve all subgraphs that match a given keyword query, we utilize a backtracking algorithm over graphs, which is described in Section 4.

Once candidate subgraphs have been retrieved, we need to rank them. We try to identify the *structured-information need* intended by the keyword query. For instance, consider the query "comedy academy award". It is likely that the user is looking for movies of genre comedy that have won the Academy Award. Thus, we need to rank the subgraphs that match this information need higher. Our ranking model is based on statistical language-models and it utilizes the distribution of terms in the whole knowledge base as a means of inferring the structured-information need of a user keyword query. For instance, given that the term 'comedy' appears often in the object of triples with predicate `hasGenre` and the terms 'academy' and 'award' appear often in the object of triples with predicate `hasWonPrize` , we can infer that the most likely structured triple-pattern query intended by the keyword query "comedy academy award" is `<?x hasGenre Comedy; ?x hasWonPrize Academy_Award>` . We thus rank subgraphs that match this implicit structured-query higher. We explain our ranking model in Section 5.

To show the effectiveness of our retrieval model, we create a benchmark for keyword queries over two real-world RDF datasets and use it to compare our ranking model to well-known IR and DB techniques for keyword search over structured data. Our evaluation results are highlighted in Section 6.

2. RELATED WORK

The work on keyword search over structured data can be classified into two classes. The first class aims at mapping the keyword query into one or more structured query. For instance, the authors in [27] 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. They then 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.

The work on query inference from a user's natural language question in [18] is also closely related. It utilizes natural-language processing tools and try 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.

The second class of work on keyword search over structured data overcomes the aforementioned issues by directly retrieving results of the keyword query. The work on keyword search over XML data for instance falls into this category. XKSearch [29] 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 [9] 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 subelements that already contain all of the query keywords. However, all these techniques assume a tree-structure and thus can not be directly applied to graph-structured data such as RDF graphs.

Also, closely related to our work is the language-modeling approach for keyword search over XML data proposed in [14]. The authors assume that a keyword query has an implicit mapping of each keyword into XML element(s). Their ranking is based on the hierarchal language-models proposed in [20] and they utilize the distribution of terms in the elements of the XML collection to weight 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 [2, 12, 10, 8] (the exception is [17] 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 [2, 13] (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 [4, 10, 17] or language models [19].

A closely related work that combines structure and content for ranking is the LM-based ranking model in [19] for ranking objects (entities in an RDF setting). 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 [3] 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.

3. SYSTEM OVERVIEW

Our knowledge base consists of a set of SPO-triples such as the one shown in Table 1. To be able to process keyword queries, we construct a virtual document for each triple t_i which we refer to as D_i. D_i contains a set of keywords that are extracted from the subject and object of the triple, and representative keywords for the predicates (these can be generated from extraction patterns for instance [25]). For example, the triple `<Innerspace hasWonPrize Academy_Award>` would be associated with the following document: {*innerspace, won, prize, academy, award*}. In case there are any textual information associated with the triple t_i such as the contextual text from which the triple was extracted [6], we can also extract all the keywords there and add them to D_i. We stem the terms in document D_i using any standard stemmer and store them in an inverted index. In addition, we also store the term frequency of term w in D_i, which we refer to as $c(w, D_i)$.

Given a keyword query, we utilize our inverted index to retrieve, for each query keyword, a list of matching triples. We then join the triples from different lists based on their subjects and objects to retrieve subgraphs with one or more triples. However, we only construct subgraphs that contain triples from *different* lists, corresponding to matches to different (sets of) keywords. The intuition behind this is that we assume that the user has a precise information need in mind that can be precisely represented using a set of triple patterns. However, since the user cannot express her information need using triple patterns, she represents each triple pattern using a set of keywords. Without any knowledge about which keywords map to which triple pattern, we need to consider all extremes: from all keywords representing a single triple pattern up to each single keyword representing an individual triple pattern. Thus, the results

Subgraphs	Keywords
Traffic hasWonPrize Academy_Award	academy, award
Innerspace hasGenre Comedy Innerspace hasWonPrize Academy_Award	comedy academy, award
Toy_Story hasGenre Comedy Toy_Story hasWonPrize Academy_Award	comedy academy, award
Road_Trip hasGenre Comedy	comedy
Diner type Comedy_films Diner hasWonPrize Academy_Award	comedy academy, award
The_Pink_Panther type Criminal_comedy_films The_Pink_Panther hasWonPrize Academy_Award	comedy academy, award
Police_Academy type Comedy_films The_Darwin_Awards type Comedy_films	academy, comedy award, comedy

Table 2: All subgraphs retrieved for the query "comedy academy award"

to the user information need would be subgraphs with one or more triples up to the number of keywords in the user query. Our algorithm for retrieving subgraphs is described in Section 4.

For example, consider the information need of finding comedies that have won the Academy Award. Furthermore assume that the user expressed this information need using the keyword query "comedy academy award". The results for such a query would then be single triples matching one or more query keywords, subgraphs with 2 triples matching at least 2 query keywords and so on. Table 2 shows all subgraphs retrieved given the query from our example RDF collection in Table 1. The second column shows the set of matched keywords by each triple in the corresponding subgraph.

Since keyword queries introduce additional ambiguity that is not present in the case of structured triple-pattern queries, result ranking becomes very crucial. For instance, the subgraphs in Table 2 differ in their size (i.e., how many triples they contain) as well as how many keywords they match. In addition, they also differ in their semantics. For instance, consider the last subgraph in Table 2. It states the fact that the movies `Police_Academy` and `The_Darwin_Awards` are both comedy movies. The rest of the subgraphs in Table 2 describe movies that have genre comedy and have won the Academy Award. Recall our earlier observation that the user keyword query is a representation of an implicit structured triple-pattern query. Thus, in order to provide an effective ranking, the system must infer what is the most likely structured query the user has in mind, and rank the subgraphs based on how well they match this implicit structured query. Our ranking model, described in Section 5, does this by combining the structure and the contents of the triples in the ranking function.

4. SUBGRAPH RETRIEVAL

As mentioned in the previous section, the first step is to retrieve the set of subgraphs that match the user keyword query. In order to avoid retrieving subgraphs that are arbitrarily long, we restrict the subgraphs retrieved to have the following two properties:

1. The subgraphs should be unique and maximal. That is, each subgraph retrieved should not be a subset of any other subgraph retrieved.

2. The subgraphs should contain triples matching different sets of keywords. That is, no triples in the same subgraph would match the exact same set of keywords. If two triples match the same set of keywords, they are parts of two different possible results to the user query, and should be considered as parts of two separate subgraphs.

Our subgraph-retrieval algorithm starts by retrieving the lists of all triples matching the query keywords. That is, given a query

Algorithm 1 RETRIEVESUBGRAPHS(E)

```
1: for each edge t ∈ E do
2:     X ← {t ∈ A(t)}
3:     EXTENDSUBGRAPH({t}, X)
4: end for
```

Algorithm 2 EXTENDSUBGRAPH(G, X)

```
1: while X ≠ φ do
2:     Remove an arbitrary chosen edge t from X
3:     if L({t}) ⊈ L(G) and L(G) ⊈ L({t}) then
4:         X' ← X ∪ {t' ∈ NEIGHBORS(t, G)}
5:         EXTENDSUBGRAPH(G ∪ {t}, X')
6:     end if
7: end while
8: if MAXIMAL(G) ≠ true then
9:     print G
10:    return
11: end if
```

$q = \{q_1, q_2, ..., q_m\}$ where q_i is a single keyword, we utilize our inverted index to retrieve lists $\{L_1, L_2, ..., L_m\}$ where L_i is the list of all triples that match the keyword q_i (see Table **??**). Let the set of all unique triples in all the lists be E. This set E can be viewed as a disconnected graph which we refer to as the *query graph* . Recall that each triple can be viewed as an edge where its subject and object are nodes.

We adapt the backtracking algorithm for network-motif detection in [28] to retrieve the subgraphs from the query graph. The modified algorithm utilizes adjacency lists for edges. Given an edge t_i from list L_i, its adjacency list $A(t_i)$ would contain all neighbor edges t_j from all other lists L_j. Two edges are considered neighbors if they share a common node. That is, given an edge t_i = <s_i p_i o_i>, an edge t_j = <s_j p_j o_j> would be a neighbor of t_i if $s_i = s_j$, $s_i = o_j$, $o_i = s_j$ or $o_i = o_j$. In order to retrieve only unique subgraphs, we associate with each edge t_i an id and we only add a neighbor to the adjacency list of t_i if its id is greater than that of t_i. Also, to ensure that we do not consider joining triples that match the same set of keywords, we only add a neighbor t_j to the adjacency list $A(t_i)$ of triple t_i if and only if $t_i \notin L_j$ and $t_j \notin L_i$. Finally, we loop over all edges and generate all unique subgraphs using the following two algorithms.

Algorithm 1 loops over all the edges and for each edge t extracts its neighbors from its adjacency list $A(t)$. Algorithm 2 takes as an input a subgraph and a list of neighbors and recursively tries to add edges to this subgraph. The condition in line 3 of Algorithm 2 ensures that only edges that belong to at least one different list other than the lists the edges of the current subgraph G belong to are considered. This ensures that we construct only subgraphs whose edges match different sets of keywords. The function $L(G)$ returns the set of lists the edges of a subgraph G belong to. Once an edge is added to the current subgraph, we also add its neighbors that are not neighbors of edges in G to the current list of neighbors and continue. The function $NEIGHBORS(t, G)$ retrieves all neighbors of an edge t that are not neighbors to edges in G. Finally, the function $MAXIMAL(G)$ ensures that the retrieved subgraph is unique and maximal.

5. RANKING MODEL

In the previous section, we presented a graph-searching algorithm that retrieves a set of subgraphs matching a given keyword query. We now explain how we rank these subgraphs. Our rank-

ing model is based on statistical language-models (LMs) [21] and works as follows. Given a query $Q = \{q_1, q_2, ..., q_m\}$ where q_i is a single term and a subgraph $G = \{t_1, t_2, ..., t_n\}$ where t_j is a triple, we rank the subgraph G based on the *query likelihood* or the probability of generating the query Q given the subgraph G's LM. Assuming independence between the query terms, the query likelihood $P(Q|G)$ is computed as follows:

$$P(Q|G) = \prod_{i=1}^{m} P(q_i|G) \quad (1)$$

where $P(q_i|G)$ is the probability of the term q_i in the LM of G. The LM of subgraph G is computed as a mixture model of the LMs of its constituent triples as follows:

$$P(q_i|G) = \sum_{j=1}^{n} \frac{1}{n} P(q_i|t_j) \quad (2)$$

That is, the probability of a term q_i in the subgraph LM is the average of its probability in the triples LMs. Note that more than one triple in subgraph G can match the same keyword q_i and thus averaging over all the triples is a natural choice. The LM of a triple t_j can then be directly computed using the document of the triple . Recall from Section 3 that each triple t_j is associated with a virtual document D_j which is composed of all the terms associated with the triple. However, this approach completely ignores the structure of the triples and treats every triple as a bag-of-words. For instance, consider the query "comedy academy award" to find comedies that have won the Academy award. Now, consider the 2 subgraphs G_1 = <Innerspace hasGenre Comedy; Innerspace hasWonPrize Academy_Award> and G_2 = <The_Darwin_Awards type Comedy_films; Police_Academy type Comedy_films> matching the query. Given the intended information need of the query, we should rank the subgraph G_1 higher.

In our ranking model, we try to take into consideration the structure of the triples as an additional evidence of how well they match the structured-information need intended by the keyword query. This is motivated by our earlier remark that we built our retrieval model on: a user keyword query is a representation of an implicit structured triple-pattern query. Considering our example query "comedy academy award", the term 'comedy' most likely refer to the triple pattern <?x hasGenre Comedy> given the fact that the term 'comedy' appears more often in the documents of triples of the form <s hasGenre Comedy> . Similarly, the terms 'academy' and 'award' would likely refer to the pattern <?x hasWonPrize Academy_Award> . Thus, it would be desirable to assign higher probability mass to triples that match these patterns (i.e., triples with predicate hasGenre for the keyword 'comedy' and hasWonPrize for the keywords 'academy' and 'award').

To this end, we set the probability of a term q_i in the triple t_j's LM or $P(q_i|t_j)$ in equation 2 to $P(q_i|D_j, r_j)$. That is, the probability of a term in a triple LM does not only depend on the document of the triple t_j, but also on its predicate which we denote by r_j. Applying Bayes' rule, we have:

$$P(q_i|D_j, r_j) = \frac{P(q_i|D_j)P(r_j|q_i, D_j)}{P(r_j|D_j)} \quad (3)$$

Furthermore, we set $P(r_j|q_i, D_j)$ as a linear combination of the following two components [24, 16, 15]:

$$P(r_j|q_i, D_j) = \beta P(r_j|q_i) + (1-\beta)P(r_j|D_j) \quad (4)$$

The first component in equation 4 is the probability that the predicate r_j is relevant to the term q_i whereas the second component is the probability that the predicate of triple t_j is r_j. The latter can be set to the extraction accuracy of triple t_j for instance. Since this value is not generally present in RDF knowledge bases, we assume that we are always fully confident in the extraction quality of any triple, and set $P(r_j|D_j)$ to 1. The parameter β is a weighting parameter that controls the effect of each component on the ranking and can be set using training queries.

Substituting equation 4 in equation 3 and simplifying, we have:

$$P(q_i|D_j, r_j) = \beta P(q_i|D_j)P(r_j|q_i) + (1-\beta)P(q_i|D_j) \quad (5)$$

where $P(q_i|D_j)$ is the probability of generating the term q_i from the triple document D_j which can be estimated using a maximum likelihood estimator after smoothing with the collection probability as follows:

$$P(q_i|D_j) = \alpha \frac{c(q_i, D_j)}{|D_j|} + (1-\alpha)\frac{c(q_i, Col)}{|Col|} \quad (6)$$

where $c(w, D_j)$ is the term frequency of term w in document D_j, $|D_j|$ is the length of document D_j (i.e., the sum of the term frequencies of all terms in D_j), Col is the whole collection constructed by concatenating all the documents of all the triples in the knowledge base and $|Col|$ is the length of the whole collection. Finally, the parameter α is a smoothing parameter and is set according to Dirichlet prior smoothing [30].

The only remaining component to estimate in equation 5 is the probability of relevance of the predicate r_j to the query term q_i. In order to estimate this probability, we first construct a document R_j for each predicate r_j in the knowledge base concatenating the documents of all the triples with such predicate. For instance, given the predicate hasGenre , we construct a document which is a concatenation of all the documents of all triples of the form <s hasGenre o> . Once we have constructed a document R_j for each predicate r_j, we set the probability of relevance of r_j to a term q_i or $P(r_j|q_i)$ in equation 5 to the $P(R_j|q_i)$ (i.e., the probability of relevance of the document R_j to the term q_i). Applying Bayes' rule, the probability $P(R_j|q_i)$ can be estimated as follows:

$$P(R_j|q_i) = \frac{P(q_i|R_j)P(R_j)}{P(q_i)} = \frac{P(q_i|R_j)P(R_j)}{\sum_k P(q_i|R_k)P(R_k)} \quad (7)$$

where $P(w|R_j)$ is the probability of generating the term w given the document R_j which is estimated using a maximum-likelihood estimator as in equation 6 and $P(R_j)$ is the prior probability of the document R_j being relevant to any term, which we set uniformly. For example, using the above technique, the probability $P(hasGenre|comedy)$ would be higher than $P(actedIn|comedy)$ given the fact that the keyword 'comedy' appears much more often in the documents of triples that have predicate hasGenre than those that have predicate actedIn .

6. EXPERIMENTAL EVALUATION

6.1 Setup

We evaluated our retrieval model using a comprehensive user-study over two RDF datasets. The first dataset was derived from the LibaryThing community, which is an online catalog about books. The second dataset was derived from the Internet Movie Database (IMDB). The data from both sources was automatically parsed and converted into RDF triples.

Recall that our retrieval model assumes that each triple t_i is associated with a document D_i. The triples documents were constructed by using keywords derived from the triple entities, and representative words for the relations. For example, the relation isMarriedTo was represented using the terms *{marry, wife, husband, spouse, etc}*. This was done manually since we did not

#entities	Example entity types	#triples	Example relations	#unique terms
LibraryThing Dataset				
48,000	`book`, `author` `user`, `tag`	700,000	`wrote`, `hasFriend`, `hasTag`, `type`	21,821
IMDB Dataset				
59,000	`movie`, `actor` `director`, `producer`, `country` , `language`	600,000	`actedIn`, `directed`, `won`, `marriedTo`, `produced`, `hasGenre`	80,584

Table 3: Overview of the datasets

have that many relations in our datasets, but for bigger datasets, the representations of relations can be generated automatically using a dictionary, or by utilizing the textual extraction-patterns in case the triples were extracted using some IE technique from free-text[25]. Once each triple was associated with a set of keywords, we stemmed all the keywords using the Stanford stemmer, removed stop words and created an inverted index over the triples. Table 3 gives an overview of the datasets.

Due to the lack of an appropriate query benchmark for *keyword search* over RDF data, we had to create a benchmark and gather relevance assessments ourselves. The queries we used for evaluation were a subset of the query benchmark in [6]. The benchmark there contains a set of structured queries, possibly augmented with keywords, along with their descriptions. We extracted *30* queries from there, 15 for each dataset and represented each query using a set of keywords. We opted for 30 queries only since we pooled 50 results per each query, and gathered relevance assessment for each result using at least 4 different human judges. Overall, we had about *15,000* unique relevance assessments for the 30 queries. All evaluation queries, results and relevance assessments are available at `http://www.mpii.de/~elbass/demo/rdftext.txt`.

We compared our ranking model, which we refer to as the *Structured LM* approach, to 3 competitors: 1) a baseline language-modeling approach (Baseline LM), 2) the Web Object Retrieval Model (WOR) [19] and 3) the BANKS system [2]. We chose these 3 competitors since they represent the family of approaches applicable to our setting, namely: keyword search over structured data. The rest of the approaches sketched in Section 2 do not directly apply to our setting and thus were omitted from our preliminary evaluation.

6.2 Relevance Assessments

For each evaluation query, we retrieved the top-50 results retrieved using each one of the 4 models described above. We then pooled all the results together and presented the set of all unique results from the pool to 13 human judges in no particular order, along with the query description. The judges were all computer-scientists in two different research institutes. For the case of results retrieved using WOR, we presented the entity name as a result and provided the judges with a link to the Wikipedia article for that entity (in case there was one) in order to help them decide whether a result is relevant or not. For the rest of the results, we just presented the subgraphs for judgment.

We asked the judges to assess the results on a 4-levels scale: 3 corresponding to results that completely match the information need as given by the query description, 2 corresponding to results that do not completely match the information need but are still highly related to it, 1 corresponding to results that do not really match the information need of the query, but the results still make sense and add valuable information to the user and finally 0 corresponding to trivial, or nonsense results. Each result was evaluated by 4 different judges. The levels of agreement between the judges as measured by the Kappa coefficient were *0.449* for the LibraryThing dataset and *0.542* for the IMDB dataset. We also computed the agreement for relevant and irrelevant results only (i.e., assuming that levels 3,2,1 are relevant and 0 is irrelevant). We obtained a Kappa coefficient of *0.397* for LibraryThing and *0.671* for IMDB which are in line with the numbers reported for standard TREC evaluation campaigns. For instance, the TREC legal track for 2006 reports a Kappa value of 0.49 on 40 queries, the opinion detection task in 2009 reports a Kappa value of 0.34, and the TREC 2004 Novelty track reports a value of 0.54 for sentence relevance.

Model	NDCG @20	NDCG @10	NDCG @5
Structured	**0.764**	**0.817**	**0.840**
BANKS	0.637	0.647	0.639
WOR	0.576	0.596	0.621
Baseline	0.397	0.368	0.351

Table 4: Average NDCG values for both datasets

Model	NDCG @20	NDCG @10	NDCG @5
Librarything			
Structured	**0.861**	**0.880**	**0.889**
BANKS	0.762	0.734	0.710
WOR	0.621	0.624	0.623
Baseline	0.395	0.361	0.333
IMDB			
Structured	**0.667**	**0.754**	**0.791**
BANKS	0.513	0.560	0.569
WOR	0.530	0.567	0.618
Baseline	0.399	0.376	0.370

Table 5: Average NDCG values with parameter learning

6.3 Evaluation Results

We conducted 3 experiments: an overall evaluation using all evaluation queries, a training experiment to set the parameters of the models that involve ones, and a cross-validation to predict how well our parameter-learning approach would generalize.

Overall Evaluation. In the first experiment, we report the average NDCG values (Normalized Discounted Cumulative Gain [11]) over all *30* evaluation queries at levels 20, 10 and 5 using all 4 different models in Table 4. The values for the Structured LM approach and the BANKS system reported in the table are the ones achieved when the models parameters were set to their optimum values (i.e., $\beta = 0.9$ for the Structured LM approach and $\lambda = 1$ for BANKS).

As can be seen from Table 4, the Structured LM approach significantly outperforms ($p - value < 0.05$ with a one-tailed t-test) all other methods in terms of NDCG values at all levels. In the next experiment, we explain how to set the models parameters using training queries.

Training Results. In the second experiment, we used one dataset for training and the other for testing. That is, the 15 queries for the IMDB dataset were used as a training set to learn the optimal parameter setting for the Structured LM approach and BANKS. The 15 queries for LibraryThing were then used to test the performance of the different methods. We did the same thing using the LibraryThing queries for training and the IMDB queries for testing. The

learning procedure was as follows. For the Structured LM approach, we computed the average NDCG at level *50* over the 15 training queries, setting the parameter β to a value between 0 and 1. We achieved the highest average $NDCG@50$ for both datasets when β was set to 0.9. For BANKS, we did the same thing using the same set of training queries and setting the parameter λ to a value between 0 and 1, and we achieved the highest average NDCG at level 50 when λ was set to 1. Table 5 shows the average NDCG values over the test queries at levels 20, 10 and 5.

Similar to the first experiment, the Structured LM approach significantly outperforms ($p - value < 0.05$ with a one-tailed t-test) all other methods in terms of NDCG values at all levels for both datasets. In order to test how well our training strategy generalizes, we performed a cross-validation experiment which we report next.

Cross-Validation Results. The third experiment was a cross-validation experiment to show how well the parameter learning procedure we described above generalizes over unseen datasets. We performed a leave-one-out cross validation, where 14 out of the 15 queries for each dataset were used as a training set to determine the the value of the parameter β, and then the left-out query was used for testing. We repeated the same process such that each evaluation query is used for validation once, and we averaged the NDCGs over all the validation queries. For BANKS, we also performed a cross-validation to validate the learning of its parameter λ, and again averaged the NDCGs over all the queries. For the IMDB dataset, the results were identical to those reported in Table 5 for all approaches, and for the LibraryThing dataset, the results were also the same as in the training experiment, except for a slight change in the case of the Structured LM approach (with NDCG values of *0.814*, *0.833* and *0.841* at levels 20,10 and 5, respectively). That is, similar to the results of the first two experiments, the Structured LM approach outperforms all other methods for both datasets.

7. CONCLUSION

We proposed a retrieval model for keyword queries over RDF graphs. Our retrieval model adopts backtracking algorithms to retrieve subgraphs matching the query keywords. Our model provides a result ranking based on a novel structure-aware language-modeling approach. We have shown through a preliminary, yet comprehensive user-study that our retrieval model outperforms well-known techniques for keyword search over structured data.

8. REFERENCES

[1] 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.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, 2002.

[3] R. Blanco, P. Mika, and H. Zaragoza. Entity search track submission by yahoo! research barcelona. SemSearch, 2010.

[4] T. Cheng, X. Yan, and K. Chen-Chuan Chang. Entityrank: Searching entities directly and holistically. In *VLDB*, 2007.

[5] A. Doan, L. Gravano, R. Ramakrishnan, and S. Vaithyanathan (Editors). Special issue on managing information extraction. *ACM SIGMOD Record*, 37(4), 2008.

[6] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum. Language-model-based ranking for queries on RDF-graphs. In *CIKM*, 2009.

[7] Freebase: A social database about things you know and love. http://www.freebase.com.

[8] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, 2008.

[9] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: ranked keyword search over xml documents. In *SIGMOD*, 2003.

[10] V. Hristidis, H. Hwang, and Y. Papakonstantinou. Authority-based keyword search in databases. *TODS*, 33(1), 2008.

[11] K. Järvelin and J. Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[12] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

[13] G. Kasneci, M. Ramanath, M. Sozio, F. M. Suchanek, and G. Weikum. Star: Steiner tree approximation in relationship-graphs. In *ICDE*, 2009.

[14] J. Kim, X. Xue, and W. B. Croft. A probabilistic retrieval model for semistructured data. In *ECIR*, 2009.

[15] O. Kurland. The opposite of smoothing: a language model approach to ranking query-specific document clusters. In *SIGIR*, 2008.

[16] O. Kurland and L. Lee. Corpus structure, language models, and ad hoc information retrieval. In *SIGIR*, 2004.

[17] G. Li, B.C. 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.

[18] V. Lopez, V. Uren, E. Motta, and M. Pasin. Aqualog: An ontology-driven question answering system for organizational semantic intranets. *Web Semant.*, 5(2):72–105, 2007.

[19] Z. Nie, Y. Ma, S. Shi, J. Wen, and W. Ma. Web object retrieval. In *WWW*, 2007.

[20] P. Ogilvie and J. Callan. Hierarchical language models for xml component retrieval. *Advances in XML Information Retrieval*, 2005.

[21] J. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, 1998.

[22] W3c: Resource description framework (rdf). www.w3.org/RDF/.

[23] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 2(1), 2008.

[24] L. Si, R. Jin, J. Callan, and P. Ogilvie. A language modeling framework for resource selection and results merging. In *CIKM*, 2002.

[25] F. Suchanek, M. Sozio, and G. Weikum. SOFIE: A self-organizing framework for information extraction. In *WWW*, 2009.

[26] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3), 2008.

[27] 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.

[28] S. Wernicke. A faster algorithm for detecting network motifs. In *WABI*, 2005.

[29] Y. Xu and Y. Papakonstantinou. Efficient keyword search for smallest lcas in xml databases. In *SIGMOD*, 2005.

[30] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR*, 2001.