Language-model-based Ranking for Queries on RDF-Graphs

Shady Elbassuoni[1] Maya Ramanath[1] Ralf Schenkel[1] Marcin Sydow[2]* Gerhard Weikum[1]
[1] Max-Planck Institute for Informatics, Saarbrücken, Germany
{elbass,ramanath,schenkel,weikum}@mpii.de
[2] Polish-Japanese Institute of Information Technology, Warsaw, Poland
msyd@pjwstk.edu.pl

ABSTRACT

The success of knowledge-sharing communities like Wikipedia and the advances in automatic information extraction from textual and Web sources have made it possible to build large "knowledge repositories" such as DBpedia, Freebase, and YAGO. These collections can be viewed as graphs of entities and relationships (ER graphs) and can be represented as a set of subject-property-object (SPO) triples in the Semantic-Web data model RDF. Queries can be expressed in the W3C-endorsed SPARQL language or by similarly designed graph-pattern search. However, exact-match query semantics often fall short of satisfying the users' needs by returning too many or too few results. Therefore, IR-style ranking models are crucially needed.

In this paper, we propose a language-model-based approach to ranking the results of exact, relaxed and keyword-augmented graph-pattern queries over RDF graphs such as ER graphs. Our method estimates a query model and a set of result-graph models and ranks results based on their Kullback-Leibler divergence with respect to the query model. We demonstrate the effectiveness of our ranking model by a comprehensive user study.

Categories and Subject Descriptors

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

General Terms

Languages, Design, Experimentation

1. INTRODUCTION

Building entity-relationship (ER) graphs has received considerable attention in the recent database, IR, and WWW literature. For example, information-extraction techniques [10, 28] have been successfully applied to textual as well as semi-structured Web sources such as Wikipedia, to build large-scale "knowledge repositories" such as DBpedia [3], Freebase [12], YAGO [31], and also community-specific collections such as DBLife [9] or Libra [25]. These repositories typically contain entities such as people, locations, movies, companies, conferences, etc. and the relationships between them such as bornIn, actedIn, hasGenre, isCEOof, isPCmemberOf, and so on. Such data conceptually forms a large graph with nodes corresponding to (typed) entities and edges denoting (typed and possibly weighted) relationships, and it can be conveniently represented in the form of subject-property-object (SPO) triples of the Semantic-Web data model RDF [27]. When triples are extracted from Web-pages, they can be associated with a variety of weights, including, extraction confidence, "witness" count (the number of times the triple was seen in the corpus), entity extraction confidence, etc.

Similar kinds of ER graphs arise in Web 2.0 settings, for example, in social-tagging communities such as `librarything.com`. Here RDF is again a convenient way of representing the wealth and diversity of data items such as user groups, friendships, annotations, ratings, etc. In contrast to traditional databases, there is not necessarily a prescriptive schema and the data providers – humans – strongly prefer a relaxed pay-as-you-go approach. Overall, RDF-style ER graphs nicely capture the entire spectrum from loose collections of data items to curated databases. Searching these kinds of data sources enables capturing underlying semantics of the data; a task often difficult to achieve with traditional Web Search.

As an example, consider a snapshot of a movies ER graph, shown in Figure 1. Entities are enclosed in boxes and the relationships between them are indicated by directed edges. The same graph can be expressed as a set of RDF triples, as shown in Table 1. We refer to an edge of the graph, together with its two nodes, as an ER fact or, equivalently, as an SPO triple.

Search on this kind of ER/RDF data is naturally expressed by

Subject (S)	Property (P)	Object (O)
Woody_Allen	directed	Match_Point
Woody_Allen	directed	Hollywood_Ending
Woody_Allen	actedIn	Hollywood_Ending
Woody_Allen	hasWonPrize	Academy_Award
Woody_Allen	hasWonPrize	BAFTA_Award
Scarlett_Johansson	actedIn	Match_Point
Tea_Leoni	actedIn	Hollywood_Ending
Match_Point	type	English_Movie
Hollywood_Ending	type	English_Movie
Vicky_Cristina_Barcelona	type	English_Movie

Table 1: A subset of RDF triples for the ER graph in Figure 1

*Work performed when the author was visiting the Max-Planck Institute for Informatics.

CIKM'09, November 2–6, 2009, Hong Kong, China.

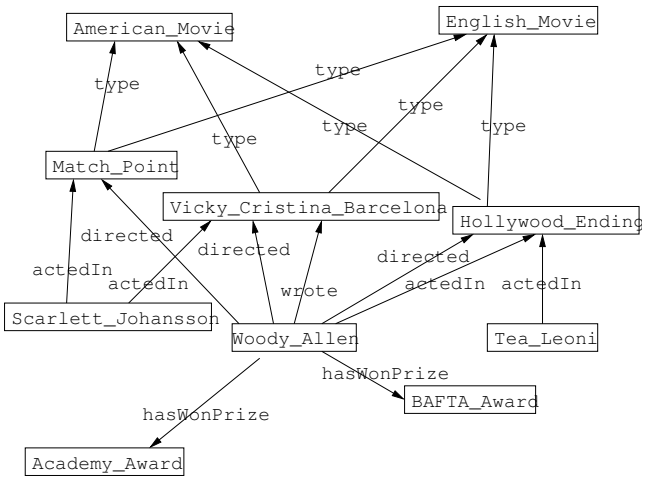

Figure 1: Example of an entity-relationship graph

Query
Woody_Allen directed ?x .
Woody_Allen actedIn ?x
Results
Woody_Allen directed Hollywood_Ending .
Woody_Allen actedIn Hollywood_Ending
Woody_Allen directed Stardust_Memories .
Woody_Allen actedIn Stardust_Memories
Woody_Allen directed Manhattan .
Woody_Allen actedIn Manhattan

Table 2: Example query and 3 unranked results

means of structured graph-pattern queries. As an example, consider a query which asks for all movies directed by Woody Allen in which he played a role. Based on a syntax similar to that of the W3C-endorsed query language SPARQL [30], we can express the query as shown in Table 2. This query consists of 2 *triple patterns* with variable names starting with a question mark. Patterns are combined by a logical conjunction, which is denoted by the dot. A result graph is a subgraph of the underlying knowledge-graph which binds the variables and matches the constants in the query.

Solely using the expressive but Boolean-match SPARQL-like languages is often too restrictive. Users prefer seeing a ranked result list rather than a list of unranked matches. For example, for the query in Table 2, well-known movies like "Hollywood Ending" or "Manhattan" should precede less-known movies like "Stardust Memories". Additionally, if the user is interested only in movies set in New York, the query could be augmented with keywords "New York" to promote, in the ranking, movies such as "Manhattan" and demote movies such as "Hollywood Ending".

Consider another example query asking for movies by Woody Allen in which both Woody Allen and Scarlett Johansson played roles. As shown in Table 3, an exact match would yield *only one* result for this query (the first result). But, there are many movies in which Woody Allen had multiple responsibilities and in which Scarlett Johansson had a role. For example, Scarlett Johansson acted in "Match Point" which Woody Allen wrote and directed. This result can be returned only when the original query is *relaxed* – that is, an approximate match to the original query is allowed. Table 3 shows examples of such approximate matches.

These considerations suggest that it is desirable to have an IR-

Query
Woody_Allen actedIn ?x .
Scarlett_Johansson actedIn ?x .
Woody_Allen directed ?x
Results
Woody_Allen actedIn Scoop .
Scarlett_Johansson actedIn Scoop .
Woody_Allen directed Scoop
Woody_Allen wrote Vicky_Cristina_Barcelona .
Scarlett_Johansson actedIn Vicky_Cristina_Barcelona .
Woody_Allen directed Vicky_Cristina_Barcelona
Woody_Allen wrote Match_Point .
Scarlett_Johansson actedIn Match_Point .
Woody_Allen directed Match_Point

Table 3: Example query and 3 exact and approximate results

style ranking model for ER-graph languages like SPARQL. Also, it should be possible to express keyword conditions together with the structured predicates of SPARQL patterns and approximate matches to the query should be allowed. This would be in analogy to prior work on XML IR which has enhanced XPath and XQuery by various forms of text-search and ranking capabilities (see, e.g., [2, 8] and references given there). However, in contrast to XML trees, we now need to address the more difficult setting of *graphs* and a potentially much higher structural and typing *diversity*. The latter also distinguishes our scope from prior work on keyword search over relational graphs (for example, [4]).

Contribution and Outline. In this paper, we investigate triple-pattern queries and show how to augment them with keyword search. This can be seen as the SPARQL counterpart of XPath-FullText. Our main contribution is a novel ranking model based on statistical language models (LMs) for exact, relaxed, and keyword-augmented queries.

LMs are the state-of-the-art foundation of modern IR [23, 33] and will be explained in Section 3. They are based on generative probabilistic models for text features in documents. Our new approach advances such models into the realm of RDF graphs and provides a seamless model for exact, relaxed and keyword-augmented graph-structured queries consisting of triple patterns. Our framework is described in section 4.

Our ranking model, described in Section 5, constructs LMs for the query and for each possible result graph, and ranks the results based on the Kullback-Leibler (KL) divergence between the query and result-graph LMs. We conducted a user study on two real-life datasets – excerpts of IMDB (`imdb.com`) and LibraryThing (`librarything.com`) – to show the quality of our search-result rankings. The results of our user study are reported in Section 7. We start by outlining related work.

2. RELATED WORK

Our work on ranking of the results to structured queries in RDF graphs is closely related to work on IR over structured data. Based on the types of data and queries handled, we classify prior work on ranking as follows: i) keyword queries on unstructured data (documents), ii) structured queries on structured data, iii) keyword queries on structured data, and iv) keyword-augmented structured queries on structured data. We make the following broad classification of ranking techniques: i) structure-based ranking and ii) content-based ranking. Many of the related works that we refer to incorporate both measures (especially in the case of XML IR). It is not possible to compare any of these techniques to our work

because of our unique setting of graph queries over a single knowledge graph , and with graphs as results (except for [20]; see discussion below). For example, some of these systems deal with graphs, but allow only keyword queries [4], others allow both structured and keyword queries, but are limited to tree patterns rather than graphs and they return trees as results [15, 1].

2.1 Keyword Queries on Unstructured Data

The main technique that we use from the standard IR literature is that of language models (LMs) and KL-divergence for result ranking [21]. An overview is given in Section 3.

In recent years, keyword querying has been carried over to the extended setting of *entity search and ranking*, also referred to as expert finding [11, 26, 29]. Here, results are named entities (e.g., companies, products, publications, authors), but the queries are still keyword-based. In most of these approaches, entities are assumed to be embedded in textual form in Web pages and other traditional kinds of documents. For the approaches that treat entities as first-class citizens [25], see Subsection 2.3 below. Extended forms of LMs and PageRank-inspired spectral analyses are used to rank the entities that qualify for a keyword query.

The key difference to our setting is that our corpus is a single redundancy-free knowledge graph instead of a set of documents, our queries are triple patterns rather than keywords and the output is a ranked list of result graphs instead of documents. We describe how to adapt language modeling techniques for this new setting.

2.2 Structured Queries on Structured Data

Ranking for structured queries has been investigated, to a small extent, for restricted forms of SQL queries. Result ranking models have been developed for selection-join queries, using either tf*idf-based models [7] or probabilistic-IR models [5] that leverage attribute-value statistics in both the database and the workload. It is not possible to carry over these techniques to our setting of schema-less and redundancy-free graph-structured data.

The closest work to this paper is the ranking model in the NAGA system [20]. NAGA introduced a query language similar to SPARQL triple patterns and used a (simpler) LM for computing a notion of informativeness. But NAGA can rank only exact matches to a given query; so the ranking is helpful only for the too-many-answers case but not for the too-few-answers problem. In contrast, our current work goes beyond this limited setting by supporting query relaxation and introducing a new notion of keyword-augmented queries.

2.3 Keyword Queries on Structured Data

The major classes of work that belong to this category include: i) keyword search on XML data which returns a ranked list of trees, ii) keyword search on graphs which returns a ranked list of Steiner trees [4, 18, 16, 13] (the exception is [22] which returns graphs), and iii) entity search on records which returns entities [25, 6].

The result ranking in each of the above is based on the *structure* of the results [4, 19] (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 [6, 16, 22] or language models [25].

Structure-based Ranking: Our work handles *structured* queries over structured data and returns graphs as results. More importantly, all the result graphs are of the *same structure* (since the input query determines the result graph structure). As a consequence, purely structure-based measures such as aggregation over the nodes and edges in the result (as in Steiner tree scoring) would not work well in our setting.

Content-based Ranking: When we deal with purely structured queries, content-based ranking is irrelevant. The reason is that both the triples in the underlying knowledge-base as well as the triple patterns in the query are crisp. That is, the subject, predicate and object in both triples and triple patterns are dealt with holistically, rather than as a set of terms. For example, it is unlikely that the same entity occurs repeatedly in the result graph unless the query demands it and hence no meaningful tf component can be associated with it. However, the situation changes when we associate each triple with keywords and allow keywords to set the context in the query. Here, content-based scoring such as tf*idf scoring is indeed relevant.

In our evaluation, we adapt an LM-based content ranking technique described in [25] to our setting and use this ranking as a competitor. Note that [25] ranks entities only, while our ranking model goes beyond this to treat triples in a holistic manner by taking into account the relationships between entities.

2.4 Keyword-augmented Structured Queries on Structured Data

XML IR like XPath Full-Text search falls into this category [15, 1]. XPath forms the tree-structured part of the query, while keyword conditions can be specified at each branch of the tree-pattern query. An important difference between XML IR and our setting is that in the former, it is possible to have results of different sizes, while in our case, the results are all of fixed structure. And so, the structure-based aspects are not as relevant to our setting as the content-based ones. The content-based ranking is again based either on tf*idf scores [1] or language models [15].

3. BACKGROUND

We start with a brief overview of the use of language models and KL-divergence in ranking results to keyword queries on text document corpora (see [14, 21] for more details).

The main intuition in the technique is to envision probabilistic processes for the generation of queries and documents. The given query and the set of documents are seen as samples from their respective distributions. For a (query, document) pair, if the underlying probability distributions are "close", then we conclude that the document is highly relevant to the query. Hence the key task is to accurately estimate a query model for the query and document models for each document and to then compare the resulting probability distributions.

The closeness between the query model and a document model is measured by the Kullback-Leibler divergence (KL-divergence), commonly used to measure the "distance" between two probability distributions:

$$KL(Q||D) = \sum_{w} Q(w) log \frac{Q(w)}{D(w)}$$

where Q and D are probability distributions corresponding to the query and document respectively and $KL(Q||D)$ measures the divergence of D from Q and the summation is over all terms w in the corpus. Note that the KL-divergence model of retrieval is a generalization of the query-likelihood method which estimates the probability of the document LM generating the query [32].

A language model estimates the probability of a term seen in the sample using the maximum likelihood estimate and smoothes it with a background model to avoid overfitting. Let w be a word in a document D (or query Q). Then, the probability of w in the document model is given by:

$$P(w) = \lambda P(w|D) + (1 - \lambda) P(w|C)$$

where $P(w|D)$ is the probability of w in the document D and $P(w|C)$ is the probability of w in the corpus C. Each of these probabilities are estimated as the fraction of occurrences of term w in the document and the corpus, respectively. For the query model, the role of C could be taken by query or click logs. The parameter λ controls the influence of the smoothing component $P(w|C)$ and is usually set empirically or by statistical learning procedures.

The topic of this paper, however, is ranking the results of graph structured queries, possibly augmented with keywords. Though the techniques we use are analogous to those described above, the fact that we support graph queries requires significant changes to the estimation procedure. The ranking model which we propose seamlessly incorporates both structured as well as keyword-augmented structured queries.

4. FRAMEWORK

Knowledge Graph. Formally, a *knowledge graph* is a labeled multi-digraph $G = \langle V, A, l_V, l_A, L\rangle$, where L is a set of labels, V is a set of nodes, $A \subseteq V \times V \times L$ a set of labeled arcs, $l_V : V \rightarrow L$ is an injective function that returns the label of a node and $l_A : A \rightarrow L$ is an injective function that returns the label of an arc such that $l_A((u, v, l)) = l$ for any $(u, v, l) \in A$. Intuitively, the nodes represent entities and arcs represent relations between them.

A knowledge graph $G = \langle V, A, l_V, l_A, L\rangle$ can be represented as a set of RDF *triples* $T(G) = \{t_1, \ldots, t_{|A|}\}$ where each triple $t_i = \langle s, p, o\rangle$ from T corresponds to exactly one arc $(u, v, l) \in A$ so that $s = l_V(u)$, $o = l_V(v)$ and $p = l_A((u, v, l))$ and represent subject s, object o, and property p. Figure 1 and Table 1 show examples of a knowledge-graph and the corresponding set of RDF triples, respectively.

Additionally, a *keyword-augmented knowledge graph* G is derived by enriching the knowledge graph with a function $K_G : A \rightarrow 2^{KEY}$ assigning a finite set of *keywords* to an arc of G (KEY is a possibly infinite set of potential keywords). Intuitively, each triple is associated with a set of keywords derived from the documents from which it was extracted.

A *witness count* $c(t)$ is associated with triple t. The witness count indicates the number of times the triple was seen and extracted from the corpus (the corpus could be the Web) and gives a measure of "importance". In addition, for each keyword w_i (text term in IR jargon) associated with t, a witness count $c(t; w_i)$ is also stored. For example, suppose that we extracted the triple Woody_Allen directed Manhattan from 30 different Web pages, so the witness count $c(t)$ for this triple is set to 30. If 20 out of these 30 contain the word "divorced", the fact-term witness count $c(t; divorced)$ is set to 20.

Each extracted triple could be associated with a confidence value, reflecting the accuracy of the employed extraction method (e.g., regular-expression matching vs. natural-language parsing vs. statistical learners) and the authenticity and authority of the data sources.

We scale down the witness count of a triple based on its confidence, by multiplying the witness count with the confidence value.

4.1 Purely Structured Graph Queries

Query. A *query* Q is a graph $Q = \langle V, A, l_V, l_A, LAB \cup VAR\rangle$, where VAR is a set of *variables* such that $LAB \cap VAR = \emptyset$. We assume that $l_V(u) \neq l_A((u, v, l)) \neq l_V(v)$ for any $(u, v, l) \in A$. If the label of any node or arc in a query is a variable we refer to its triple representation as a *triple pattern*.

For example, the following *graph* query asking for a married couple and a movie that they both have acted in consists of three triple patterns: ?x marriedTo ?y . ?x actedIn ?m . ?y actedIn ?m.

Query Result. Let $B = \langle V, A, l_V, l_A, LAB\rangle$ be a knowledge graph and $Q = \langle V', A', l_{V'}, l_{A'}, LAB \cup VAR\rangle$ a query. We will say that a subgraph $D = \langle V'', A'', l_V, l_A, LAB\rangle$of B *matches* the query Q if there exists a bijective function $f : V'' \rightarrow V'$ such that the conjunction of the following conditions holds: i) *entity match*: $l_V(v) = l_{V'}(w)$ or $l_{V'}(w) \in VAR$, for any $v \in V''$ and $w \in V'$ and ii) *relation match*: $l_A(a'') = l_{A'}(a')$ or $l_{A'}(a') \in VAR$ for any $a'' \in A''$ and $a' \in A'$. The *result* to the query Q is the set of subgraphs of B that match the query.

4.2 Keyword-augmented Structured Queries

Example Query	Woody_Allen produced ?x . Woody_Allen directed ?x
Example Query with keywords	Woody_Allen produced ?x*{murder lover}* . Woody_Allen directed ?x
Relaxed Queries	Woody_Allen ?y ?x*{murder lover}* . Woody_Allen directed ?x
	Woody_Allen produced ?x*{murder}* . Woody_Allen ?y ?x
	Woody_Allen produced ?x . Woody_Allen ?y ?x
	Woody_Allen ?z ?x . Woody_Allen ?y ?x
Extreme Query	?w ?z ?x . ?w ?y ?x

Table 4: Query Framework

A *keyword-augmented query* allows keywords to be associated with one or more triple patterns in the query.

For example, the query Woody_Allen directed ?x*{murder lover}* could be issued to ask for movies which Woody Allen directed and which had murder and lovers as a theme (top-ranked results would include Match Point and Manhattan Murder Mystery). As another example, consider the following query with two triple patterns: ?x hasGenre Comedy*{academy award}* . ?y directed ?x. This query could be issued to retrieve all comedies that have been mentioned with the terms "academy award" (e.g., were nominated for it or have won it), along with their directors.

Different triple patterns can have different keywords associated with them. For example, if we wanted to retrieve pairs of actors who played roles of detective and gangster, respectively, in the same movie, we could express this query as: ?x actedIn ?m*{police detective}* . ?y actedIn ?m*{gangster}* with different keywords for each triple pattern. Note that it is not possible to express this query with a single global keyword condition.

4.3 Relaxed Queries

Query relaxation, in our setting, alleviates the problem of "too few results" and increases recall by allowing for approximate matching of queries. For example, consider a structured query asking for movies which were both directed and produced by Woody Allen: Woody_Allen directed ?x . Woody_Allen produced ?x.

The above query would return a single answer (there is only one movie which was both produced and directed by Woody Allen). However, there are many more movies in which Woody Allen had more than one responsibility. For example, he directed and wrote Vicky Cristina Barcelona, he directed and acted in Scoop, etc. But none of these results is a match for the original query. The only way to retrieve these results is to run relaxed versions of the original

query. For example, in addition to the original query, we could formulate a relaxed query as follows: Woody_Allen directed ?x . Woody_Allen ?y ?x.

The above query keeps the first triple pattern intact – that is, the movies returned should have been directed by Woody Allen – but the second pattern is relaxed and only requires that Woody Allen be somehow connected to the movie.

There is some prior work on query relaxation for structured queries (for example, [1, 34], but they are neither comprehensive nor directly applicable to our setting). We also note that the focus of this paper is on proposing a ranking model which seamlessly encompasses exact, relaxed and keyword-augmented graph-structured queries and so, a thorough investigation of query relaxation techniques is beyond the scope of this work. We use the following simple, yet useful relaxation model to illustrate our ranking techniques.

Generating Relaxed Queries. We *relax* a triple pattern by replacing one or more of its constants with a variable. For example, the query Woody_Allen produced ?x*{new york}* can be relaxed to Woody_Allen ?y ?x*{new york}* by replacing the relation produced by a variable. The keywords themselves could be "relaxed", by choosing a subset of keywords or no keywords to remain in the query. The previous example could be relaxed to Woody_Allen ?y ?x by removing all keywords, in addition to replacing produced by a variable. Clearly, the larger the number of keywords and constants in the pattern, the larger the number of possible relaxations. Extending this to the entire query, a relaxed query contains one or more relaxed patterns. It is possible to systematically generate a number of relaxed queries from the given query by choosing all possible triple-pattern relaxations.

Weighting Relaxed Queries. A relaxed query can be weighted, relative to the original query from which it is derived. This is useful when the actual query execution considers both the original and the relaxed query and integrates query results from both.

We use the number of relaxations (that is, the number of constants substituted with variables) in a query to weight the query. The larger the number of relaxations, the lower the weight. This implies that the original query gets the highest weight, and relaxed queries with the same number of relaxed queries get equal weight.

Table 4 summarizes our querying framework (note that only a subset of relaxed queries are shown in the table). The final relaxation, termed "extreme", simply consists of two patterns of variables only. However, valid results would bind the first entities of both patterns as well as the second entities of both patterns to the *same* values, while the relation could be bound to different constants. Extreme relaxations could be useful especially in combination with keyword conditions, as a means of keyword-only search. An exact match to any of the relaxed queries is considered to be an approximate answer to the original query.

5. RANKING MODEL

Our ranking model estimates a query LM P_Q for the generation of query Q and a result-graph LM P_G for the generation of a result graph G (a subgraph of the knowledge base B). The result graphs are ranked in increasing order of the KL divergence between the query LM and the result-graph LM. The KL divergence between the two gives a measure of relevance of G with respect to Q.

We make two distinctions in our setting as compared to traditional keyword queries on documents. First, there is no notion of a document in our setting. Instead, we have a large graph of facts from which subgraphs can be constructed. And so, result units are constructed on-the-fly depending on the query. More specifically, if a query contains n-triple patterns, then any graph with n triples (an n-triple tuple) is considered to be a potential result. In the general case, we need not restrict the number of triples to n. However, we enforce this restriction since we are interested only in exact-structure matches (which will naturally contain n triples) to both original and relaxed queries.

Second, queries are made up of triple patterns, while results are made up of triples. A probability distribution over triple patterns is incomparable to a probability distribution over triples. We need to overcome this "vocabulary gap" in order to compare the query and result-graph LMs. We do this through a notion of "query instantiation". This is described in the next subsection.

5.1 Query Language Model

#	**Triple** (t_i)	$c(t_i)$
t_1.	Spielberg directed Schindler's_List	200
t_2.	Spielberg directed Munich	50
t_3.	Spielberg produced Men_in_Black	20
t_4.	Spielberg directed Jaws	50
t_5.	Tarantino directed Kill_Bill	100
t_6.	Jaws hasGenre Thriller	100
t_7.	Schindler's_List hasGenre War	20
t_8.	Munich hasGenre War	200
t_9.	Men_in_Black hasGenre Comedy	150
t_{10}.	Kill_Bill hasGenre Thriller	30

Table 5: Small knowledge base with triples and witness counts

$\hat{q_1}$	$P_Q(t)$	$\hat{q_2}$	$P_Q(t)$
t_1	200/300	t_6	100/500
t_2	50/300	t_7	20/500
t_4	50/300	t_8	200/500
		t_9	150/500
		t_{10}	30/500

Table 6: Query LM estimation for a query with two patterns. q_1 = Speilberg directed ?x . q_2 = ?x hasGenre ?y

Let $Q = \{q_1, ..., q_n\}$ be a query with n triple patterns where q_i is a triple pattern. An *instantiation* of a triple pattern q_i is a match to the triple pattern q_i. A triple pattern can have multiple instantiations. Let $\hat{q_i}$ denote the set of instantiations of q_i.

We define the query LM as a probability distribution over n-tuples of the form $T = \{t_1, ..., t_n\}$, where t_i is a triple. Assuming independence among triples,

$$P_Q(T) = \prod_i P_Q(t_i) \tag{1}$$

where $P_Q(T)$ is the probability of the n-tuple in the query LM and $P_Q(t_i)$ is the probability of triple t_i.

We now describe how to estimate $P_Q(t_i)$ in the cases of exact and relaxed queries as well as keyword-augmented queries.

Exact Matching. Let $\hat{q_i}$ be the set of triples which match the triple pattern q_i and $c(t_i)$ be the number of witnesses of triple t_i. The probability $P_Q(t_i)$ is then estimated as follows:

$$P_Q(t_i) = \begin{cases} \frac{c(t_i)}{\sum_{t \in \hat{q_i}} c(t)} & \text{if } t_i \in \hat{q_i} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

As an example, consider the small knowledge base shown in Table 5. It shows the triples as well as their witness counts. Now consider a query with two patterns: Spielberg directed ?x. ?x hasGenre ?y asking for all movies directed by Spielberg and their genres. The query is factorized into two patterns, as shown in Table 6, and each pattern is instantiated. The probabilities of the triples in the instantiation are then calculated based on their witness counts and the sum of witness counts in the instantiation set. The probabilities computed in this way are then used in Equation 1 to calculate n-tuple probabilities in the query model.

Keyword Augmentation. Probability estimation for keyword-augmented queries proceeds analogously to the procedure described above, with a few subtle differences. Each triple in the knowledge base is augmented with a term (keyword) (see Section 4) and the same triple is repeated with different terms if it has multiple terms that need to be associated with it. $c(t_i; w_k)$ is the number of witnesses for triple t_i occurring with term w_k.

We now need to compute the probability of keyword-augmented triple patterns of the form $q_i = q[w_1, ..., w_m]$ where q is a simple triple pattern and w_k is an associated keyword.

As before, we need to estimate $P_Q(t_i)$, the probability of a triple t_i in the query LM. However, since now we have a context, in the form of terms $w_1, ..., w_m$, we actually need to estimate the probability $P_Q(t_i|w_1, ..., w_m)$. That is, the probability of the triple t_i, given the context $w_1, ..., w_m$. Assuming independence between keywords, we calculate the triple probability as

$$P_Q(t_i|w_1, ..., w_m) = \prod_{k=1}^{m} [\alpha P_Q(t_i|w_k) + (1 - \alpha)P(t_i)] \quad (3)$$

where $P_Q(t_i|w_k)$ is the probability of t_i given the single-term context w_k, $P(t_i)$ is the smoothing component, and the parameter α controls the influence of smoothing. Note that it is crucial to smooth the probabilities, since otherwise $P_Q(t_i|w_1, ..., w_m) = 0$ if t_i is not associated with at least one term w_k in the context. We use uniform smoothing (i.e, a uniform probability distribution for $P(t_i)$). Now, in order to estimate the first component of Equation 3, let $\hat{q_i}$ be the set of triples which match the triple pattern q. Then,

$$P_Q(t_i|w_k) = \begin{cases} \frac{c(t_i; w_k)}{\sum_{t \in \hat{q_i}} c(t; w_k)} & \text{if } t_i \in \hat{q_i} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

With the use of Equation 3 to compute the triple probabilities, we no longer need to explicitly relax keywords in the query. For example, for the triple pattern Woody_Allen directed ?x*{murder lover}*, we need not generate the following relaxations: Woody_Allen directed ?x*{murder}*, Woody_Allen directed ?x*{lover}*. This is because the smoothing component automatically takes care of the case when one or more keywords are not present with the triple.

We use Equation 4 in Equation 3. The latter is then used in Equation 1 to calculate the n-tuple probabilities.

Relaxed Queries. As mentioned in Section 4, our ranking model is applicable to any kind of relaxation scheme. In general, let q_i^0 be the original triple pattern and let q_i^j be a relaxation.

We now need to compute the probability of a triple t_i in the query LM. Intuitively, we should weigh t_i higher if it is a match for the original triple pattern, and lower if it is a match for a relaxed version. In addition, we need to make distinctions among the relaxed pattern versions – that is, if t_i is a match to a relaxation with only one constant replaced with a variable, then it should be weighted higher than if it were a match to a relaxation with two constants replaced with variables. With these aspects in mind, we estimate the probability of t_i in the query LM as a mixture model:

$$P_Q(t_i) = \lambda_0 P_Q^0(t_i) + \lambda_1 P_Q^1(t_i) + ... + \lambda_j P_Q^j(t_i) \quad (5)$$

where $P_Q^j(t_i)$ denotes the probability of t_i for the relaxation q_i^j, λ_i weighs the contribution of each component and $\sum_i \lambda_i = 1$. $P_Q^j(t_i)$ is computed according to Equation 4 or Equation 2 depending on whether or not keywords are part of the triple pattern. In general, the λ_i are set based on the "closeness" of the relaxed pattern to the original one. In our case, we compute the λ's based on the number of relaxations in the pattern, as described in Section 4.3.

5.2 Result-Graph Language Model

The main complexity of our technique is in the estimation of the query LM. In contrast, the estimation of the result-graph LM is straightforward. For query Q with n triple patterns, we are interested in potentially relevant results consisting of n triples. We only rank results that match the structure of the original query or one of its relaxations. Given a potential result graph $T = \{t_1, t_2, ..., t_n\}$, we estimate its language model P_G. Analogously to the query LM, P_G is a probability distribution over n-tuples where

$$P_G(T) = \beta P(T|G) + (1 - \beta)P(T|B)$$

and the parameter β controls the influence of smoothing. $P(T|G)$ is 1 if $G = T$ and 0 otherwise (since the result graph G contains only the n-tuple T). For the smoothing component, we assume independence between triples:

$$P(T|B) = \prod_{i=1}^{n} P(t_i|B)$$

where $P(t_i|B)$ is estimated given the entire knowledge base and its entirety of witnesses, and is equal to:

$$P(t_i|B) = \frac{c(t_i)}{\sum_{t \in B} c(t)}$$

For example, consider the relaxed query Q = Spielberg ?r ?x (of the original query Spielberg directed ?x) and consider the single triple (1-tuple) T = Spielberg produced Men_in_Black as a possible result graph for the relaxed query. $P(T|B)$ is set to $20/920$ according to the knowledge base of Table 5 (independently of Q).

5.3 Result Ranking

Given the query LM and result-graph LMs for all n-tuples T, we now compute the KL-divergence between the query LM P_Q of query Q and the result-graph LM P_G of result G as follows:

$$KL(Q||G) = \sum_i P_Q(T_i) log \frac{P_Q(T_i)}{P_G(T_i)}$$

The result graphs are then returned to the user in ascending order of KL-divergence.

6. IMPLEMENTATION

6.1 RDF Repository

We use an Oracle11g database as the storage back-end for triples. A single table stores all triples and witness counts with the following schema: `Triples(id, arg1, relation, arg2,`

Triples				
id	**arg1**	**relation**	**arg2**	**#witnesses**
1	Brad_Pitt	actedIn	Se7en	3,210,000
2	Brad_Pitt	actedIn	Babel	483,000
3	Brad_Pitt	hasWon	Saturn_Award	354,000
4	Woody_Allen	directed	Manhattan	857,000
5	Manhattan	hasGenre	Comedy	266,000
...	...	...	...	...

Table 7: Sample Database: Triples Table

Keywords				
tid	**term**	**stem**	**tf**	**#witnesses**
1	serial	seria	4	217,000
1	killer	kill	7	483,000
1	cops	cop	9	60,000
4	york	york	2	535,000
4	divorce	divorc	4	103,000
...	...	...	...	...

Table 8: Sample Database: Keywords Table

`witness-count)`. The keywords associated with each triple are parsed and stored in a separate table with the following schema: `Keywords(tid, term, stem, tf, witness-count)`, where *tid* is the triple id in table `Triples`, *term* is the keyword as present in the witness from which the triple was extracted, *stem* is the stemmed version of the keyword, *tf* is the number of times the keyword occurred in the witness, and *witness-count* is the number of witnesses from which the triple was extracted and contained the keyword. An example database is shown in Tables 7 and 8. Indices are created on each column and beneficial column combinations.

6.2 Query Processing

All graph queries are translated into SQL to be evaluated over the database. Each query involves a set of self-joins that utilizes the indices on the `Triples` table to run efficiently. For example, consider the query to find married couples who have acted in the same movie: ?x marriedTo ?y . ?x actedIn ?m . ?y actedIn ?m. This query is translated to SQL over the `Triples` table as follows:

```
SELECT T1.arg1, T1.arg2, T2.arg2
FROM TRIPLES T1, TRIPLES T2, TRIPLES T3
WHERE
T1.relation = 'marriedTo' AND
T2.relation = 'actedIn' AND
T3.relation = 'actedIn' AND
T1.arg1 = T2.arg1 AND
T2.arg2 = T3.arg2 AND
T1.arg2 = T3.arg1
```

The SQL statement representing the query is executed and the list of matching tuples are retrieved. These tuples represent the set of candidate results that need to be ranked. Each tuple is broken into its constituting triples, which are then used to construct the query LM as described in Section 5. Additionally, the query is relaxed, and for each relaxed query the same procedure is applied. Finally, the overall query LM is computed as a mixture model of the original query LM and all relaxed queries LMs. Once the query LM is constructed, the KL-divergence between each matching tuple and the constructed query LM is computed and the tuples are ranked in ascending order according to the KL-divergence values.

In general, the query processing times ranged from a few seconds to tens of seconds for queries with more than 3 triple-patterns. Our analysis showed that the running time was mainly dominated by the DB execution time, which is highly dependent on the underlying storage system. This calls for exploring other alternatives to store the triples. For example using the RDF-specific RDF-3X engine [24] which supports efficient SPARQL query processing (note that it does not support ranking).

7. EXPERIMENTAL EVALUATION

7.1 Setup

To evaluate the effectiveness of our ranking model, we conducted a comprehensive user study over two datasets using the Amazon Mechanical Turk service. The first dataset was derived from a subset of the Internet Movie Database (IMDB) and the second dataset was derived from the LibaryThing community, which is an online catalog and forum about books. The data from both sources was automatically parsed and converted into RDF triples. Since each triple is present only once in both data sources, we had to estimate the witness count for the triples. In order to do so, we relied on the Web corpus. We issued queries to a major search engine, with the subject and object of the triple as keywords, and set the witness count to the number of hits returned by the search engine.

In addition, each triple was also augmented with keywords derived from the data source it was extracted from. In particular, for the IMDB dataset, all the terms in the plots, tag-lines and keywords fields were extracted, stemmed and stored with each triple. For the LibraryThing dataset, since we did not have enough textual information about the entities present, we retrieved the books' Amazon descriptions and the authors' Wikipedia pages and used them as sources of keywords for the triples. An overview of the datasets is given in Table 9.

#entities	Some Entity Types	#triples	Some Relationship Types
IMDB Dataset			
59,000	movie, actor director, producer country , language	600,000	actedIn, directed hasWonPrize, marriedTo produced, hasGenre
LibraryThing Dataset			
48,000	book, author user, tag	700,000	wrote, friendOf hasTag, type

Table 9: Overview of the datasets

7.2 Evaluation Queries

We used 2 different sets of evaluation queries. The first set consisted of pure structured queries that ranged from simple single-pattern queries to complex multi-pattern graph queries. We constructed *16 queries* for the IMDB dataset and *8 queries* for the LibraryThing dataset. A subset of the evaluation queries used and the corresponding number of triple patterns for both datasets are shown in Table 10.

The second set consisted of keyword-augmented queries. Again, we constructed *16 queries* for the IMDB dataset and *8 queries* for the Librarything dataset. The queries were structured queries associated with one or more keywords. A subset of the evaluation queries used and the corresponding number of triple patterns for both datasets are shown in Table 11. The keywords shown in square brackets were associated with the appropriate triple patterns.

For all queries, we retrieved both exact and approximate matches and aggregated them using the weighting scheme described in Section 5. The relaxed queries were generated using the simple tech-

nique of replacing one or more constants in the original query with a variable.

IMDB	
Structured Query Q	$\|Q\|$
An academy awarded movie produced in Australia	2
A thriller movie and its director	2
A married couple who acted in the same movie	3
An academy award for best director winner directing an Academy award for best actor winner in a movie	4
A family movie produced in the year 1995 and an actor/actress that played a role in that movie and also acts in comedies	5
LibraryThing	
Structured Query Q	$\|Q\|$
A mystery and thriller book and its writer	2
A classic 20th century book and its writer	3
A writer that writes both fiction and non-fiction books and one book from each category	4
A series book and its writer	2
A book and its writer	1

Table 10: Subsets of Purely Structured Queries

IMDB	
Keyword-augmented Query Q	$\|Q\|$
A comedy movie about [weddings]	1
An award winning actor/actress that acted in a movie that has something to do with [Spielberg]	2
An award winning director who directed a movie that is based on a [true story]	2
A romance movie produced in the year 2001 and has something to do with [Paris]	2
An academy award for best actor winner who acted in a movie with an academy award for best actress winner and the movie has something to do with [love] or [relationship]	4
LibraryThing	
Keyword-augmented Query Q	$\|Q\|$
A book that has something to do with [wizards] and has a sequel and its writer	2
A book that has something to do with [Spain] and its writer	1
A classic book and its writer and either the book or the writer won a [Pultizer] award	2
A fiction book about magic and its writer and the book has won some [award]	3
A classic book that has something to do with [revolutions] and its writer	3

Table 11: Subsets of Keyword-augmented Queries

7.3 Competitors

We compared our approach against a number of competitors that represent state-of-the-art methods in ranking over structured data. First, we compared against the work done on web object retrieval (WOR) in [25] since they use language models in order to rank results. This covers a class of competitors that deal with term-frequency based ranking of results to keyword queries on structured data. Second, we compared to the class of rankers that utilize graph properties to rank results to keyword queries over structured data by adapting the Steiner weight scoring as used in the BANKS system [18]. Finally, we compared to the closest work to ours, the language-model-based ranking in the NAGA system [20]. We describe how we adapted each competitor to rank graph results to structured queries, possibly augmented with keywords, in the following. We also describe how we handle approximate matching whenever applicable.

WOR: The key difference between WOR and our ranking approach is that we support ranking of graph results and treat triples as first class citizens in our ranking model, while the objective of WOR is to rank *entities*. Another key distinction is that WOR operates on keyword queries only.

The technique assumes that an entity is associated with a set of records extracted from web sources. This set of records forms a "document" for the entity. The relevance of such a "document" (and correspondingly, the entity associated with it) to a keyword query is estimated using LMs. That is, the probability of the document (correspondingly, the entity) generating the query is estimated based on term-frequencies and the results are returned in decreasing order of probabilities to the user.

We adapted the WOR techniques for ranking entities and extended them to rank graphs as follows. We treated triples as records and for a given entity, all triples which contain that entity formed the "document" for that entity. Given a structured query, whether keyword-augmented or not, we retrieved all result graphs matching the structure of the query. Since WOR supports keyword queries only, we converted our evaluation queries into terms and computed the rank of a result graph as the product of the probability of generating the bound entities in the graph given the query terms.

BANKS: The BANKS system enables keyword-based 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 and what nodes it connects. 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.

This directly applies to our setting. Given a query, whether purely structured or keyword-augmented, we retrieved all result graphs that matched the query or one of its relaxed versions. We then ranked the result graphs based on a combination of edge weights and node weights. The edges were weighted by the witness count of the triple (edge), and nodes were weighted based on the average in-degree of the node.

NAGA: In NAGA, the result graphs are ranked based on their likelihood of generating the query triple patterns. This probability was estimated using 3 different measures: 1) confidence of the result, 2) compactness of the result and 3) informativeness of the result. The first 2 components do not play a role in our setting, since all our triples were extracted using the same method, and thus all have the same confidence values. Compactness does not play a role in the ranking as well, since all result graphs are of the same size which is determined by the number of triple patterns in the query. Thus, the only component that affects the ranking is the informativeness component which we computed using the witness counts. The major difference between our ranking method and that of the NAGA system is in the parameter estimation technique. Moreover, NAGA does not support approximate matching and keyword-augmented queries.

7.4 Metrics

For the user study, we pooled the top-10 results obtained from each technique (including ours) and presented them to the evaluators in random order. Each result was evaluated by 7 anonymous users on the Amazon Mechanical Turk service. The evaluators were required to indicate whether the result was "highly relevant", "relevant", "somewhat relevant", "undecidable", "irrelevant" or "wrong". To measure the ranking quality of each technique, we used the Discounted Cumulative Gain (DCG) [17], which

is a measure that takes into consideration the rank of relevant documents and allows the incorporation of different relevance levels. DCG is defined as follows

$$DCG(i) = \begin{cases} G(1) & \text{if i = 1} \\ DCG(i-1) + G(i)/log(i) & \text{otherwise} \end{cases}$$

where *i* is the rank of the result within the result set, and *G(i)* is the relevance level of the result. We set $G(i)$ to a value between 0 and 5 depending on the evaluator's assessment. For each result, we averaged the ratings given by all evaluators and used this as the relevance level for the result. Dividing the obtained DCG by the DCG of the ideal ranking we obtained a *Normalized DCG (NDCG)* which accounts for the variance in performance among queries.

7.5 Results

Purely structured queries with relaxation				
Dataset	**OWN**	**WOR**	**BANKS**	**NAGA**
IMDB	**0.880**	0.751	0.777	0.798
LT	**0.876**	0.787	0.721	0.869
Keyword-augmented queries with relaxation				
Dataset	**OWN**	**WOR**	**BANKS**	**NAGA**
IMDB	**0.884**	0.722	0.782	0.776
LT	**0.853**	0.835	0.690	0.782

Table 12: Avg. NDCG for all evaluation queries

The results of our user evaluation are shown in Table 12. The reported NDCG values were averaged over all evaluation queries. In the case of purely structured queries, our ranking model outperforms all competitors for both datasets. In particular, for the IMDB dataset, we achieved more than 17% significant gain in NDCG over WOR with a one-tailed paired t-test p-value of 0.047, and more than 13% over BANKS with a p-value of 0.004. Similarily, for the Librarything dataset, we achieved more than 11% gain in NDCG over WOR with a p-value of 0.078 and more than 21% over BANKS with a p-value of 0.013.

The effectiveness of our ranking model is especially visible in the case of keyword-augmented queries. Our ranking approach again outperforms all competitors for both datasets. For the IMDB dataset, We achieved a gain of more than 22% over WOR with a p-value of 0.020 and 13% over BANKS with a p-value of 0.002 and similarily, for the librarything dataset.

Analysis. We outperformed WOR since it supports only entity ranking, and thus does not take into consideration the relations between the results. It also supports keyword queries only, and the ranking is based on term-frequencies. This is in contrast to our approach of treating a triple holistically (using witness counts), rather than as a set of terms. This indicates that when the objective is to rank graphs, it is better to treat triples holistically, rather than as a combination of entities.

For BANKS, we adapted their technique to our setting, but their ranking still depends on the static properties of the result graphs. Even weighting edges using our witness counts proved to be insufficient to deliver better quality results. And the fact that our system has a clear-cut ranking model for exact+relaxed queries as opposed to BANKS that does not, resulted in better results with our approach.

Our closest competitor in terms of setting and technique is NAGA. Even though, NAGA returns a ranked list of result graphs to structured queries, it supports neither approximate matching nor keyword-augmentation. Thus, while many purely structured queries had similar result lists, when testing our technique as a whole with relaxation, NAGA's techniques failed to give effective results. Similarly, the lack of explicit support for keywords in NAGA resulted in less effective ranking for keyword-augmented queries.

Examples. In Table 13 we show some example evaluation queries over the IMDB dataset. For each query, the top-4 results returned by our own approach, as well as the 3 other competitors are given. Due to space limitation, we just show the bound entities (i.e., matches to the variables in the queries). Next to each result, we also show the average relevance value given by the evaluators (column titled *Rel.*). Recall that each result was given a relevance value between 0 and 5, with 5 corresponding to highly relevant results.

For query $Q1$ asking for thriller movies, the top-4 results returned by our approach (OWN) are all well-known movies, as compared to both WOR and BANKS. The results returned by BANKS are not even thriller movies. This is due to the fact that even though BANKS ranking can be adapted to approximate matches, the way approximate and exact matching are later aggregated only depends on the static properties of the result graphs (i.e., edges and nodes weights).

The top-4 results returned by NAGA are the same as the ones returned by our approach since both methods rely on witness counts to estimate ranking probabilities. The superiority of our method over NAGA is more clear in the case of keyword-augmented queries and in the case where there are not enough exact matches to the given query. For example, query $Q2$ (augmented with keywords "true story") asks for movies based on a "true story" and directed by an award winning director. Our top-4 results are all movies based on a true story, which is not the case in any of the other 3 result lists returned by the competitors.

Query $Q3$ illustrates the benefit of query relaxation. The query asks for movies produced in Australia and that won an academy award. In our dataset, there is only one exact match to this query, namely, "Secrets of the Heart". For lack of space, we only show the top results for our own approach and for BANKS. We also show the award the movie won and the production country, which are bound entities for the relaxed queries. Recall that a query is relaxed by replacing one of the constants (i.e., either Australia or Academy_Award by a variable). Our method ranks the exact match at the top, while approximate matches which includes movies which have won an Academy_Award, but produced in another country, or movies produced in Australia but have won other awards are ranked lower. This is in contrast to the list returned by BANKS.

8. CONCLUSIONS

This paper has investigated ranking for a new brand of structured queries on RDF graphs. Our model and algorithms are applicable to all kinds of RDF databases, enabling ranked retrieval for SPARQL and paving the way for a new form of SPARQL-FullText extension. Our approach makes extensive use of IR methodology, specifically, statistical language models, leveraging it in the realm of structured and semistructured graph data. The presented user-study demonstrates the potential of our method. And our query relaxation techniques showed significant gains in the NDCG metric.

Our future work includes investigating graph-query relaxations more thoroughly and improving our query processing efficiency to return top-k results.

9. REFERENCES

[1] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and content scoring for xml. In *VLDB*, 2005.

Q1	A thriller movie and its director			
Rank	**OWN**	**Rel.**	**WOR**	**Rel.**
1	Batman_Begins, Christopher_Nolan	4	Deadly_Intruder, John_McCauley	3.43
2	Murder!, Alfred_Hitchcock	4	Robotix, Wally_Burr	3
3	Spider-Man_3, Sam_Raimi	3.71	Like_Minds, Gregory_J._Read	2.71
4	Eyes_Wide_Shut, Stanley_Kubrick	3.86	Khoey_Ho_Tum_Kahan, Ajab_Gul	2.86
Rank	**BANKS**	**Rel.**	**NAGA**	**Rel.**
1	-30-, Jack_Webb	2.86	Batman_Begins, Christopher_Nolan	4
2	Kill!, Kihachi_Okamoto	2.57	Murder!, Alfred_Hitchcock	4
3	If...., Lindsay_Anderson	2	Spider-Man_3, Sam_Raimi	3.71
4	Hit!, Sidney_J._Furie	2.71	Eyes_Wide_Shut, Stanley_Kubrick	3.86
Q2	**An award winning director who directed a movie that is based on a [true story]**			
Rank	**OWN**	**Rel.**	**WOR**	**Rel.**
1	Martin_Scorsese, Good_fellas	3.14	Joel_Lamangan, Babangon_Ako't_Dudurugin_Kita	0
2	Steven_Spielberg, Schindler's_List	3.43	Tracy_Seretean, Big_Mama	3.14
3	Peter_Jackson, Heavenly_Creatures	3.29	Ben_Burtt, The_American_Gangster	2.71
4	Bernardo_Bertolucci, Little_Buddha	3.19	David_Frankel, Why_We_Fight	3.29
Rank	**BANKS**	**Rel.**	**NAGA**	**Rel.**
1	Baz_Luhrmann, Australia	3.29	Clint_Eastwood, Million_Dollar_Baby	2.86
2	Woody_Allen, Vicky_Cristina_Barcelona	3.57	Eddie_Murphy, Harlem_Nights	2.71
3	Christopher_Nolan, Batman_Begins	2.14	Robert_Altman, Health	3
4	Mel_Gibson, Braveheart	3	Mel_Gibson, Braveheart	3
Q3	**An academy awarded movie produced in Australia**			
Rank	**OWN**	**Rel.**	**BANKS**	**Rel.**
1	Secrets_of_the_Heart, Academy_Award, Australia	3.71	Chiranjeevi, Padma_Bhushan, India	1.57
2	Before_Sunset, Academy_Award, USA	2.57	Three_Seasons, Independent_Spirit_Award, UK	1.43
3	Innerspace, Academy_Award, USA	1.86	Charlie_Chaplin, Academy_Honorary_Award, India	2
4	Beneath_Clouds, Australian_Film_Institute_Award, Australia	3.29	Separate_Tables, Academy_Award, USA	1.71

Table 13: Examples of IMDB queries and top-ranked results

[2] S. Amer-Yahia and M. Lalmas. Xml search: languages, inex and scoring. *SIGMOD Record*, 35(4), 2006.

[3] S. Auer, et. al. Dbpedia: A nucleus for a web of open data. In *ISWC/ASWC*, 2007.

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

[5] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic information retrieval approach for ranking of database query results. *ACM Trans. on Database Syst.*, 31(3), 2006.

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

[7] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD*, 1998.

[8] W. B. Croft and H.-J. Schek. (Eds.). Special issue on database and information retrieval integration. *VLDB J.*, 17(1), 2008.

[9] 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.

[10] A. Doan, L. Gravano, R. Ramakrishnan, and S. Vaidyanathan. (Eds). Special issue on managing information extraction. *SIGMOD Record*, 37(4), 2008.

[11] H. Fang and C. Zhai. Probabilistic models for expert finding. In *ECIR*, 2007.

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

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

[14] D. Hiemstra. *Using Language Models for Information Retrieval*. PhD thesis, University of Twente, Enschede, 2001.

[15] D. Hiemstra. Statistical language models for intelligent XML retrieval. In *Intelligent Search on XML Data*, 2003.

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

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

[18] V. Kacholia, et.al. Bidirectional expansion for keyword search on graph databases. In *VLDB*, 2005.

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

[20] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *ICDE*, 2008.

[21] J. D. Lafferty and C. Zhai. Document language models, query models, and risk minimization for information retrieval. In *SIGIR*, 2001.

[22] 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.

[23] X. Liu and W. B. Croft. Statistical language modeling for information retrieval. In *Annual Review of Information Science and Technology 39*, 2004.

[24] T. Neumann and G. Weikum. RDF-3X: a RISC-style engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

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

[26] D. Petkova and W. Croft. Hierarchical language models for expert finding in enterprise corpora. *Int. J. on AI Tools*, 17(1), 2008.

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

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

[29] P. Serdyukov and D. Hiemstra. Modeling documents as mixtures of persons for expert finding. In *ECIR*, 2008.

[30] W3c: Sparql query language for rdf. www.w3.org/TR/rdf-sparql-query/.

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

[32] C. Zhai. Statistical language models for information retrieval: A critical review. *Foundations and Trends in IR*, 2(3), 2008.

[33] C. Zhai and J. D. Lafferty. A risk minimization framework for information retrieval. *Inf. Process. Manage.*, 42(1), 2006.

[34] X. Zhou, J. Gaugaz, W.-T. Balke, and W. Nejdl. Query relaxation using malleable schemas. In *SIGMOD*, 2007.