Searching RDF Graphs with SPARQL and Keywords

Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Gerhard Weikum

Max-Planck Institute for Informatics
Saarbruecken, Germany
E-mail: {elbass,ramanath,schenkel,weikum}@mpi-inf.mpg.de

Abstract

The proliferation of knowledge-sharing communities like Wikipedia and the advances in automated information extraction from Web pages enable the construction of large knowledge bases with facts about entities and their relationships. The facts can be represented in the RDF data model, as so-called subject-property-object triples, and can thus be queried by structured query languages like SPARQL. In principle, this allows precise querying in the database spirit. However, RDF data may be highly diverse and queries may return way too many results, so that ranking by informativeness measures is crucial to avoid overwhelming users. Moreover, as facts are extracted from textual contexts or have community-provided annotations, it can be beneficial to consider also keywords for formulating search requests. This paper gives an overview of recent and ongoing work on ranked retrieval of RDF data with keyword-augmented structured queries. The ranking method is based on statistical language models, the state-of-the-art paradigm in information retrieval. The paper develops a novel form of language models for the structured, but schema-less setting of RDF triples and extended SPARQL queries.

1 Motivation and Background

Entity-Relationship graphs are receiving great attention for information management outside of mainstream database engines. In particular, the Semantic-Web data model RDF (Resource Description Format) is gaining popularity for applications on scientific data such as biological networks [14], social Web2.0 applications [4], large-scale knowledge bases such as DBpedia [2] or YAGO [13], and more generally, as a light-weight representation for the "Web of data" [5].

An RDF data collection consists of a set of subject-property-object triples, SPO triples for short. In ER terminology, an SPO triple corresponds to a pair of entities connected by a named relationship or to an entity connected to the value of a named attribute. As the object of a triple can in turn be the subject of other triples, we can also view the RDF data as a graph of typed nodes and typed edges where nodes correspond to entities and edges to relationships (viewing attributes as relations as well). Some of the existing RDF collections contain more than a billion triples.

As a simple example that we will use throughout the paper, consider a Web portal on movies. Table 1 shows a few sample triples. The example illustrates a number of specific requirements that RDF data poses for querying:

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Subject (S)	Property (P)	Object (O)
Ennio_Morricone	hasType	Composer
Ennio_Morricone	hasWonPrize	Academy_Award
Ennio_Morricone	composedSoundtrack	C'era_una_volta_il__West
Henry_Fonda	actedIn	C'era_una_volta_il__West
Henry_Fonda	hasWonPrize	Academy_Award
Claudia_Cardinale	actedIn	C'era_una_volta_il__West
Claudia_Cardinale	hasWonPrize	Golden_Berlin_Bear
El_Laberinto_del_Fauno	hasType	Movie
El_Laberinto_del_Fauno	hasTag	Fantasy
El_Laberinto_del_Fauno	hasTag	Franco_Regime_in_Spain
Guillermo_del_Toro	directed	El_Laberinto_del_Fauno
Guillermo_del_Toro	hasWonPrize	Academy_Award
Javier_Navarette	createdFilmMusic	El_Laberinto_del_Fauno
Javier_Navarette	nominatedFor	Grammy_Award

Table 1: SPO triples in the RDF data model

1. Despite the repetitive structure in some parts of the data, there is often a high diversity of property names across the entire dataset. Thus, we need a flexible query language to search and explore *schema-less* data. The W3C-endorsed SPARQL language offers the equivalent of select-project-join queries, but in contrast to SQL, allows wildcards for property names.

2. While this is a powerful feature for schema-less data, it would still be a Boolean-match evaluation. Therefore, SPARQL queries may often produce empty results when queries are too specific, or overwhelm the user or application program with huge result sets. This calls for meaningful *ranking* of search results, based on statistics about the data and user queries.

3. While ranking is desired for structured query conditions, we should also consider that RDF triples often come with additional textual components. For example, movie portals also contain user comments, knowledge bases that are automatically built by information extraction from Web sources should also include the source texts, and biological data collections are often annotated by expert users or may reference PubMed articles. To tap on these potentially valuable texts, an RDF query language should also support text search with *keywords* and *phrases*.

This paper gives an overview on how these three requirements can be addressed. It is based on recent and ongoing research on IR-style ranking for RDF data [8, 9]. In Section 2, we discuss how to extend the SPARQL language with keyword search. In Section 3, we show how to compute principled rankings for search results in a schema-less but structured data setting. In Section 4, we sketch experimental studies, as evidence for the viability of the proposed model. We conclude with an outlook on open issues.

2 Extending the SPARQL Query Language

The SPARQL language is the W3C standard for querying RDF data. A query consists of conjunctions of elementary SPO search conditions, so-called *triple patterns*. For example, a question about prize-winning composers who have composed music for movies that feature prize-winning actors, with results consisting of composer-

movie pairs, can be expressed in SPARQL as follows:

Select ?x, ?m *Where* { ?x hasType Composer . ?x hasWonPrize ?p .
?x composedSoundtrack ?m . ?m hasType Movie .
?a actedIn ?m . ?a hasWonPrize ?q }

Each triple pattern has one or two of the SPO components replaced by variables such as ?x , the dots between the triple patterns denote logical conjunctions, and using the same variable in different triple patterns denotes join conditions. For the example data of Table 1, this query returns Ennio Morricone and C'era Una Volta il West (which stars Henry Fonda). However, the query would miss results about movies whose directors won an award or with actors or directors nominated for an award (which often is a major honor already). Moreover, as there is no prescriptive schema for property names, there are a variety of ways for expressing the appearance of musical compositions in movies; referring only to the composedSoundtrack property is unduly restrictive. Fortunately, SPARQL allows relaxing the query by having variables for property names. This way, the query could be re-phrased into the more liberal form:

Select ?x, ?m *Where* { ?x hasType Composer . ?x hasWonPrize ?p .
?x ?p ?m . ?m hasType Movie . ?a ?r ?m . ?a hasWonPrize ?q }

This is still a highly structured query, with typed entity variables, join conditions, etc. But it is now so relaxed that we would obtain a large number of results on a realistically sized movie collection. This too-many-results situation thus calls for a ranked result list, in the IR spirit, rather than a result set merely based on Boolean matching. We will discuss our approach to ranking in the next section.

As mentioned before, RDF triples may have associated text passages, e.g., the text from which a fact was automatically extracted or user-provided comments from a Web2.0 community. In this case, that text provides extra information that we can consider in formulating queries. For example, suppose we search for film-music composers who have also written classical music and whose compositions appear in dramatic movie scenes such as gunfights in westerns or battles in easterns. This is difficult if not impossible to express by structured query conditions alone. To overcome this problem, we would prefer to simply specify keywords – but keywords within the context of a more precise triple pattern:

Select ?x, ?m *Where* { ?x hasType Musician {*classical, composer*} .
?x composedSoundtrack ?m . ?m hasType Movie {*gunfight, battle*} }

Here, two of the triple patterns are augmented by keyword conditions. The query should return results such as Tan_Dun composedSoundtrack Hero. Similarly, we could ask for rock musicians whose music is used in love scenes in movies (as mentioned in descriptions of the movie plot or discussed in fan communities), or we could refine our earlier condition about awards by specifying that the award citation should include certain keywords, and so on.

It is important to note that the query still has a rich structure with several triple patterns and that the association of keywords with individual triple patterns is crucial. The keyword parts have to be matched by facts whose associated text contains the specified keywords. This imposes a *semantic grouping* on the total set of query keywords. Without this structure, we would get different, possibly inappropriate, matches: for example, movies about classical composers such as Mozart with music composed by rock stars that were involved in gunfights and their personal battle with drugs (to give an outrageous example).

The outlined approach is close in spirit to languages like XQuery Full-Text [1] - except, and this is crucial, that we are dealing with RDF-style graph data and not with XML trees. The semantic grouping of keywords is impossible to express with today's Internet search engines. Separate phrase conditions such as "classical composer" and "gunfight battle" return very different results based on exact matching of consecutive words, and would miss good results such as "composer of the opera . . . a masterpiece for classical orchestra".

3 Ranking with Statistical Language Models

Whenever queries return many results, we need ranking based on the *informativeness* of the answers. Users prefer prominent entities and salient facts as answers. For example, the query about award-winning composers for movies with award-winning actors or directors should return prominent results such as "El Laberinto del Fauno" or "Hero" rather than matches such as "The Muse" (which features music by Grammy winner Elton John but is not a well-known movie).

State-of-the-art ranking models in IR are based on *statistical language models*, *LMs* for short. They have been successfully applied to passage retrieval for question answering, cross-lingual search, ranking elements in semistructured XML data, and other advanced IR tasks [7]. An LM is a generative model, where a document d is viewed as a probability distribution over a set of words $\{t_1, ..., t_n\}$ (e.g., a multinomial distribution), and a query $q = \{q_1, ..., q_m\}$ with several keywords q_i is seen as a sample from this distribution. The parameters of the distribution d are determined by maximum-likelihood estimators in combination with advanced smoothing (e.g., Dirichlet smoothing). This is not unlike $tf * idf$-style frequency-based models, but LMs are more principled and the smoothing method is often decisive for good results. Now one can estimate the likelihood of query q for different candidate documents, and the one document that maximizes this likelihood should be the highest-ranked result. Alternatively, we can also associate queries with LMs: a query is a probability distribution over keywords, derived from the query itself and, for example, a general user-interest profile or other sources for smoothing. Then, the query-likelihood model is equivalent to ranking based on the Kullback-Leibler divergence (KL-divergence for short) of the query LM with regard to the candidate document LMs [16]. KL-divergence is an information-theoretic measure (aka. relative entropy) for comparing two probability distributions. It is always non-negative and zero only for identical distributions.

3.1 Entity-Relationship Ranking

Recently, extended LMs have been developed for entity ranking in the context of expert finding in enterprises and Wikipedia-based retrieval and recommendation tasks [10–12, 15]. These models are limited to entities – the nodes in an entity-relationship graph. In contrast, general knowledge search needs to also consider the role of relations – the edges in the graph – for answering more expressive classes of queries. Moreover, the discussed work on entity IR is still based on keyword search and does not consider structured query languages like SPARQL. Ranking for structured queries has been intensively investigated for XML [1], and, to a small extent, for restricted forms of SQL queries [6]. Ranking has also been studied in the context of keyword search on relational graphs (e.g., [3]). However, these approaches do not carry over to the graph-structured, largely schema-less RDF data collections and the expressive queries discussed before.

What we need for RDF knowledge ranking is a generalization of entity LMs that considers relationships (RDF properties) as first-class citizens. Recent work on the NAGA search engine [8, 9] has addressed these issues and developed full-fledged LM-based methods for ranking the results of extended SPARQL queries.

In IR, typical LM-based ranking for documents, passages, or entities are associated with informative distributions over words (or phrases or N-grams) and queries have straightforward LMs (essentially just the words in the query). In contrast, our LMs for structured RDF search are probability distributions over facts (RDF triples). Our approach is to construct a query LM for the query and result LMs for each potential result. The KL-divergence between the query LM and a result LM gives a measure of relevance with the results being presented to the user in increasing order of KL-divergence.

3.2 Estimating the Query LM

The LM for a triple pattern, the basic unit of a structured query, considers all possible substitutions of its variables by matching triples from the underlying RDF data collection. Consider the pattern ?x directed ?y. This is

satisfied by all director-movie pairs from directed triples in the data. The LM for this triple pattern is a probability distribution over these triples (with smoothing by giving a small amount of probability mass to all other triples).

The probability of a given triple in the LM of a triple pattern can be viewed as the probability that the user is interested in this particular triple as an answer to her question. As discussed above, some triples are more informative than others, because they refer to important directors, important movies, etc. We can incorporate this aspect into the LM by using statistical weights for different triples in order to construct a non-uniform distribution. To this end, we consider the *witnesses* of a given triple: how often, on the Web or in the news, do we see this triple. Alternatively, if we construct the RDF data collection by automatic information extraction from Web sources, the witnesses of a triple are the distinct sources from which we have extracted the triple. In our implementation, we issued keywords queries for each triple against a major search engine and used the reported result sizes as estimates for witness counts. These counts are pre-computed and stored in indexes.

#	**Triple** (t_i)	$c(t_i)$	$P_Q(t_i)$
t_1	Ivana_Baquero actedIn El_laberinto_del_fauno	200	$200/1000 = 0.2$
t_2	Henry_Fonda actedIn C'era_una_volta_il_West	250	$250/1000 = 0.25$
t_3	Holly_Hunter actedIn The_Piano	200	$200/1000 = 0.2$
t_4	Robert_Duvall actedIn Apocalypse_Now	350	$350/1000 = 0.35$

Table 2: Triples matching ?a actedIn ?m with witness counts

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:

$$P_Q(t_i) = \frac{c(t_i)}{\sum_{t \in \hat{q_i}} c(t)}$$

For the triple pattern ?a actedIn ?m, Table 2 shows a very simple LM over four actedIn triples (smoothing over all triples is ignored here for simplicity).

Now consider a query with two or more patterns, for example, [?x directed ?y . ?x hasWonPrize Academy_Award] asking for movies whose directors have won the Academy Award. For queries with N triple patterns, we now define the query LM to be a probability distribution over N-tuples of triples. For tractability of both parameter estimation and query-time computation, we assume probabilistic independence among pairs of triple patterns and compute the entire query LM as:

$$P_Q(T) = \prod_i P_Q(t_i)$$

where $P_Q(T)$ is the probability of the N-tuple $(t_1, \ldots, t_N)$ in the query LM and $P_Q(t_i)$ is the probability of triple t_i. $P_Q(t_i)$ is computed as previously described.

The Query LM for Keyword-Augmented Structured Queries. For estimating the query LM of keyword-augmented queries, each triple in the RDF collection is conceptually extended by an associated keyword taken from textual annotations or witness documents. The same triple is repeated with different words if it has multiple words associated with it. $c(t_i; w_k)$ is the number of witnesses for triple t_i occurring with word 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 words $w_1, ..., w_m$, 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

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

where $P_Q(t_i|w_k)$ is the probability of t_i given the single-keyword 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 word w_k in the context. We use uniform smoothing (i.e, a uniform probability distribution for $P(t_i)$). Now, let $\hat{q_i}$ be the set of triples which match the triple pattern q. Then,

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

The necessary count values $c(t_i; w_k)$ are precomputed and stored in indexes (indexed on triple identifier and keyword identifier).

3.3 Result LM and Ranking

For query Q with N triple patterns, let G be a result. Now, the LM of G, denoted P_G, is estimated over all N-tuples as: $P_G(T) = \beta P(T|G) + (1 - \beta)P(T|C)$, where β is another smoothing parameter. If G is the N-tuple T, then $P(T|G) = 1$, otherwise, $P(T|G) = 0$. For the smoothing component, probabilistic independence is assumed: $P(T|C) = \prod_{i=1}^{N} P(t_i|C)$ where $P(t_i|C)$ is estimated given the entire corpus or data collection: $P(t_i|C) = \frac{c(t_i)}{\sum_{t \in KB} c(t)}$

Given the query and candidate-result LMs, the KL-divergence between the query LM P_Q of query Q and a result LM P_G of result G is computed as follows:

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

The results are returned to the user in ascending order of KL-divergence.

4 Experimental Studies

We conducted a user study with relevance assessments collected on the Amazon Mechanical Turk platform. Our dataset was derived from a subset of the Internet Movie Database (IMDB) and consisted of around 600,000 RDF triples. In addition, each triple was augmented with keywords derived from the data source where it was extracted from. In particular, all the terms in the plots, tag-lines and keywords fields were extracted and stored with each triple.

We constructed 10 structured queries and 10 keyword-augmented queries. We presented the top-10 results for each query to 7 different evaluators who judged the results based on a 6-point scale ranging from "highly relevant" (5) to "irrelevant" (0). To measure the ranking quality, we used the Discounted Cumulative Gain (DCG) [7], which is a measure that takes into consideration the rank of relevant documents and allows the incorporation of different relevance levels. 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.

Table 3 shows the 10 evaluation queries and their NDCG values for the case of purely structured SPARQL queries. On average, we achieved a high NDCG value of 0.88. Similarly, Table 4 shows the 10 keyword-augmented SPARQL queries we used to evaluate the quality of the result ranking in the case where the queries contained keyword conditions. The third column gives the corresponding NDCG value for each query. On average, we obtained an NDCG value of 0.866. For both types of queries, the NDCG results are excellent (1.0

would be perfect). Note that the keyword-augmented queries cannot be expressed at all in purely structured form. If we dropped the keyword conditions and merely used the structural conditions, we would obtain much weaker if not useless results.

ID	SPARQL Query	NDCG
1	?a actedIn ?m . ?a hasWonPrize ?p . ?m hasWonPrize Academy_Award	0.834
2	?a hasWonPrize ?p . ?a actedIn ?m1 . ?a actedIn ?m2	0.891
3	?d directed ?m . ?d actedIn ?m	0.901
4	?a1 isMarriedTo ?a2 . ?a1 actedIn ?m . ?a2 actedIn ?m	0.905
5	?d hasWonPrize Academy_Award_for_Best_Director . ?d directed ?m . ?a actedIn ?m . ?a hasWonPrize Academy_Award_for_Best_Actor	0.869
6	?m hasGenre Comedy . ?a actedIn ?m . ?a directed ?m	0.956
7	?m hasGenre Comedy . ?a1 actedIn ?m . ?a2 actedIn ?m	0.894
8	?a hasWonPrize Academy_Award_for_Best_Actress. ?a actedIn ?m . ?a diedOnDate ?t	0.912
9	?d directed ?m . ?d hasWonPrize ?p1 . ?m hasWonPrize ?p2	0.818
10	?m1 hasGenre Family . ?m1 hasProductionYear 1995 . ?a actedIn ?m1 . ?m2 hasGenre Comedy . ?a actedIn ?m2	0.821

Table 3: NDCG for Purely Structured Queries

ID	SPARQL Query	NDCG
1	?a actedIn ?m {spielberg} . ?a hasWonPrize ?p	0.745
2	?m hasGenre Comedy {christmas} . ?a1 actedIn ?m . ?a2 actedIn ?m	0.846
3	?a1 hasWonPrize Academy_Award_for_Best_Actor . ?a1 actedIn ?m {relationship} . ?a2 hasWonPrize Academy_Award_for_Best_Actress . ?a2 actedIn ?m {love}	0.873
4	?d hasWonPrize Academy_Award_for_Best_Director . ?d directed ?m {new, york} . ?a actedIn ?m . ?a hasWonPrize Academy_Award_for_Best_Actor	0.872
5	?m hasGenre Comedy {school, friends}	0.939
6	?m hasGenre Comedy {police} . ?a actedIn ?m . ?a directed ?m	0.933
7	?d directed ?y {true, story} . ?d hasWonPrize ?p	0.921
8	?a hasWonPrize Academy_Award_for_Best_Actress . ?a actedIn ?m {paris} . ?a diedOnDate ?t	0.920
9	?d directed ?m {love} . ?d hasWonPrize ?p1 . ?m hasWonPrize ?p2	0.849
10	?m1 hasGenre Family {wedding} . ?m1 hasProductionYear 1995 . ?a actedIn ?m1 . ?m2 hasGenre Comedy {serial, killer} . ?a actedIn ?m2	0.762

Table 4: NDCG for Keyword-augmented Queries

In Table 5 we show some example purely-structured queries. For each query, the top-5 results are given. Due to space limitations, 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 *Rel.*). Recall that each result was given a relevance value between 0 and 5, with 5 corresponding to highly relevant results.

For Query 3 in Table 3, asking for a movie directed and acted in by the same person, the top-5 results are all well-known movies (see Table 5). Similarly, for Query 5 in Table 3, asking for a director who won an Academy award and directed a movie that an Academy-awarded actor acted in, we also obtain very prominent movies, directors, and actors.

For the case of keyword-augmented SPARQL queries also, our result ranking is superior. For instance, consider Query 2 in Table 4, asking for comedy movies about "Christmas" and their directors. The top-5 movies

Q. ID	Rank	Result	Rel.
3	1	Sylvester_Stallone, Rambo	3.29
	2	Mel_Gibson, Braveheart	3.71
	3	Clint_Eastwood, Million_Dollar_Baby	3.71
	4	Woody_Allen, Annie_Hall	3.71
	5	Ben_Stiller, Zoolander	4.14
5	1	Martin_Scorsese, Robert_De_Niro, Casino	3.43
	2	Milo_Forman, Jack_Nicholson, One_Flew_Over_the_Cuckoo's_Nest	3.29
	3	William_Wyler, Gregory_Peck, Roman_Holiday	3.29
	4	Robert_Zemeckis, Tom_Hanks, Forrest_Gump	3.00
	5	Sydney_Pollack, Dustin_Hoffman, Tootsie	2.86

Table 5: Top-ranked results for some example purely-structured queries

Q. ID	Rank	Result	Rel.
2	1	Miracle_on_34th_Street, Maureen_O'Hara, Edmund_Gwenn	2.86
	2	Love_Actually, Liam_Neeson, Alan_Rickman	3.14
	3	Bad_Santa, John_Ritter, Billy_Bob_Thornton	2.86
	4	Jingle_All_the_Way, James_Belushi, Arnold_Schwarzenegger	3.29
	5	Christmas_in_Connecticut, Sydney_Greenstreet, Barbara_Stanwyck	3.14
4	1	Martin_Scorsese, Robert_De_Niro, Mean_Streets	3.29
	2	Elia_Kazan, Marlon_Brando, On_the_Waterfront	3.29
	3	James_L._Brooks, Jack_Nicholson, As_Good_as_It_Gets	3.14
	4	Spike_Jonze, Nicolas_Cage, Adaptation.	3.29
	5	John_Schlesinger, Dustin_Hoffman, Midnight_Cowboy	3.57

Table 6: Top-ranked results for some example keyword-augmented queries

(see Table 6) are all well-known comedies that have christmas as a theme. Query 4 in Table 4 asks for a director that won an Academy award, and directed a movie that an Academy-awarded actor acted in, and in addition the movie should be related to "New York". The top answers, shown in Table 6, are all salient movies featuring New York.

5 Future Directions

In our current work, the text-search conditions are limited to keywords. However, IR systems usually offer a richer repertoire of predicates: phrase matching (i.e., several contiguous keywords), proximity search (i.e., several non-contiguous keywords within short distance), negated conditions (i.e., taboo words that must not appear in a result), query expansion (i.e., adding related words that are not explicitly given in the query), and more. Our LM-based framework is geared for this, and can compute meaningful rankings even for such richer queries with a structured "backbone". However, efficient indexing and query processing pose major challenges.

Personalizing and diversifying search results are also open issues in the extended-SPARQL setting. For example, consider a question about Oscar winners from Europe. In the standard setting, we would rank people like Bruce Willis, Anthony Hopkins, Roman Polanski, or Ingrid Bergman as top results. But suppose the user has previously expressed strong interest in music, e.g., by browsing information on contemporary composers. Then the personalized search result should prefer people like Ennio Morricone, Hans Zimmer, or Javier Navarrete (all of whom won Oscars for film music). Now consider that the query should also return salient movies by these European Oscar winners. Returning only movies with Ennio Morricone's music on the top ranks would be rather

monotone, even if this is the user's favorite composer. Instead, it is highly desirable to have more diversity in the top-ranked results.

The querying and ranking models presented in this paper have shown very good results so far. The next step will be to extend our framework to address the above open issues.

References

[1] S. Amer-Yahia, M. Lalmas: XML Search: Languages, INEX and Scoring. SIGMOD Record 35(4), 2006

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, Z. G. Ives: DBpedia: A Nucleus for a Web of Open Data. ISWC/ASWC 2007

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, S. Sudarshan: Keyword Searching and Browsing in Databases using BANKS. ICDE 2002

[4] J. Breslin, A. Passant, S. Decker: The Social Semantic Web, Springer, 2009

[5] C. Bizer, T. Heath, T. Berners-Lee: Linked Data - The Story So Far. To appear in: International Journal on Semantic Web and Information Systems (IJSWIS), 2010.

[6] S. Chaudhuri, G. Das, V. Hristidis, G. Weikum: Probabilistic Information Retrieval Approach for Ranking of Database Query Results. ACM Trans. Database Syst. 31(3), 2006

[7] W. B. Croft, D. Metzler, T. Strohman: Search Engines - Information Retrieval in Practice. Addison-Wesley, 2009

[8] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, G. Weikum: Language-model-based Ranking for Queries on RDF-Graphs. CIKM 2009

[9] G. Kasneci, F. Suchanek, G. Ifrim, M. Ramanath, G. Weikum: NAGA: Searching and Ranking Knowledge. ICDE 2008

[10] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, W.-Y. Ma: Web Object Retrieval. WWW 2007

[11] D. Petkova, W. B. Croft: Hierarchical Language Models for Expert Finding in Enterprise Corpora. ICTAI 2006

[12] P. Serdyukov, D. Hiemstra: Modeling Documents as Mixtures of Persons for Expert Finding. ECIR 2008

[13] F. Suchanek, G. Kasneci, G. Weikum: YAGO: a Core of Semantic Knowledge. WWW 2007

[14] UniProt: Universal Protein Resource, `http://www.uniprot.org/`

[15] D. Vallet, H. Zaragoza: Inferring the Most Important Types of a Query: a Semantic Approach. SIGIR 2008

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