Keywords and RDF Fragments: Integrating Metadata and Full-Text Search in Beagle++

Tereza Iofciu, Christian Kohlschütter, Wolfgang Nejdl, Raluca Paiu

L3S Research Center / University of Hanover
Deutscher Pavillon, Expo Plaza 1
30539 Hanover, Germany
{iofciu,kohlschuetter,nejdl,paiu}@l3s.de

1 Introduction

Full-text search engines and metadata repositories have so far investigated very different approaches to search, mainly due to their separate and different storage systems for information and data. As we have argued in previous papers, though, integrating full-text and metadata search capabilities is crucial for powerful semantic desktop search systems [3]. Semantic metadata is able to represent important contextual information on the desktop and provide both more research results (better recall) as well as more sophisticated ranking (better precision) compared to simple full-text search.

Recently, integrating databases and information retrieval technologies has been discussed by different research groups. [1] motivates the need for systems which integrate DB and IR querying capabilities, especially in the case of searching in libraries, where librarians are forced to switch between full-text and database search functionalities. An interesting approach to querying semi-structured data such as XML has been presented in [2], which introduces the notion of XML fragments to be used as parts of queries, inspiring part of the work described in the current paper. A system building on RDF is Magnet, which is presented in [4]. It supports naïve-user navigation of structured information via a domain-independent search framework and user interface. Magnet can be used both for browsing RDF as well as XML collections of data.

In our context we want to achieve integrated full-text and metadata search on the desktop including two important functionalities present in current search engines or IR systems, namely the possibility to return results which only partially match the query and to rank results accordingly. In contrast to this, usual database queries return all results as an unranked set of hits.

2 Searching with RDF Fragments

2.1 The Vector Space Model

For conducting full-text search, it is common to use the well-known Vector Space Model (VSM). An important aspect of this model is that documents and queries have the same structure. This means that, when searching full-text document collections, the queries are *compared to full-text documents, just as two documents are.* The results are then ranked according to the *similarity* of the documents with the query. The relevance of a document d for a specific query q is computed by using a similarity function such as the cosine measure:

$$\rho(q, d) = \frac{\sum_{t \in q \cap d} w_q(t) * w_d(t)}{\|q\| * \|d\|} \tag{1}$$

The similarity $\rho(q, d)$ is computed in a t-dimensional space, t being the number of different terms appearing in all documents. $w_x(t)$ stands for the weight of term t in x, x representing the document or the query. This can be boolean (0 = not in document, 1 = contained) or, for instance, defined by the TF·IDF measure. The latter normalizes the impact of frequent/infrequent words in a document per division of a term's occurrences in a specific document by the number of overall documents the term is contained in. Looking closer at this similarity function, the reader can notice that results do not necessarily have to include all query terms. If they do not include at least one, though, the similarity will be zero. In contrast to the Euclidean distance, in the cosine similarity function only those terms influence the similarity value which occur in *both* vectors.

2.2 Searching both Full-Text and Metadata

How do we generalize the vector space model to a semantic desktop search scenario? The key idea here is to consider extended documents which consist both of full-text and of metadata, and allow the user query for these documents in an integrated way. Figure 1 shows such an extended document, with an RDF graph representing the metadata associated to the document.

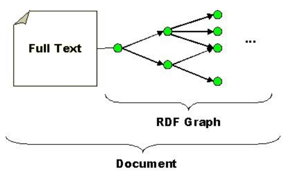

Fig. 1. A searchable document, consisting of full-text and reachable metadata

A query containing full-text and RDF metadata properties can then be expressed in the same way as a small document, including both keywords and RDF fragments. This approach offers a very easy-to-use query language. For example, if we consider the natural-language query: *"Return the papers which I received from Bob via an email attachment, considering RDF indexing"*, this can be expressed by the following RDF fragment query:

“`RDF indexing`” */storedFrom/attachedTo/receivedFrom* `Bob`.

If Alice remembers that the paper Bob sent to her has been presented at a VLDB conference, she can use this additional information to refine her query. So, the query now is *“Return the papers which I received from Bob via email, considering RDF indexing, and that have been presented at VLDB”*, which is represented as an RDF fragment query as follows:

“`RDF indexing`” */storedFrom/attachedTo/receivedFrom* `Bob`.
/presentedAt `VLDB`.

It is useful to support approximate and imprecise metadata queries as well. For example, in the previous query, Alice could have erred on the name of the conference. So articles presented at SIGIR, which matched the keyword and *receivedFrom* restrictions, should be also returned as hits, though with a smaller rank. In addition, we should also return appropriate documents if a metadata path has only been specified incompletely (for example, */receivedFrom* `Bob`).

2.3 Semantics of RDF Fragment Queries

Besides providing an intuitive and integrated way to query for full-text and metadata, we also want to be able to efficiently support indexing and searching on the desktop using a single VSM-based full-text search engine like Lucene. Based on the vector space model, we will then consider a document a potential result if it has a non-null similarity with the corresponding query. In addition to the basic vector space model functionalities, we also support additional restriction operators commonly used in full-text search engines: *MUST*, *MUST NOT* and *PHRASE* operators. Just like any other term, a phrase can be prefixed with the *MUST* or *MUST NOT* operator sign (`+` or `-`), to force its appearance/absence, or be enclosed in double quotes for the PHRASE operator to force a list of keywords to appear consecutively. If a term/phrase is not signed, its appearance is desired but not inevitable (i.e. an implicit *SHOULD* operator).

If we compare these query capabilities to relational query languages, our query model supports the operators selection and projection, as well as union, intersection and set difference. The only operator not supported is the join, which however only precludes queries such as ‘Find papers from authors who cite themselves’, seldomly thought of by search engine users.

3 Indexing and Querying of Full-text and Metadata

3.1 Background: Lucene and Beagle Indexing

We are building our search infrastructure on Beagle which is a GNOME project for indexing and searching resources on the desktop. Beagle uses Lucene as high performance full-text search engine, which is a multi-purpose information retrieval library for adding indexing and search capabilities to various applications. Lucene can index and search any data that can be converted to a textual representation.

At the core of all full-text search engines is high performance indexing, i.e. processing the original data into highly efficient cross-reference lookup tables in order to facilitate rapid searching. A common way of implementing the vector space model is to store terms in inverted indices, associating them with the documents they are contained in as well as including the corresponding TF and IDF values.

3.2 Indexing Metadata to Support RDF Fragment Queries

The default approach to index metadata as used in Beagle is to define one field per metadata predicate; full-text and document URIs are also stored as fields. This idea is not suitable for metadata paths. Adding one field per path is not possible as the paths can contain a lot of different predicates, determined only at runtime. Furthermore, the ability to rank documents by metadata literals is lost because the TF·IDF measure used by the vector space model does not span across fields.

Our approach uses one single "metadata" field for all metadata associated to a document. For each directly associated statement, we store both predicate and object of the statement as text in this field. This makes storage independent of any underlying RDF schema, as the predicates are represented as terms rather than field names. To describe more complex contexts, we store *predicate paths*. These paths represent the properties which are not directly annotated to the document, but can be reached by following a path in the RDF graph starting from the document node via subject–predicate–object connections. RDF fragments can be matched using phrase queries on the metadata field, and results are returned almost instantaneously, as querying reduces to a lookup in the Lucene index.

4 Experiments

Our test dataset consisted of 150 publications, with metadata extracted from the CiteSeer database and some additional one regarding email information. We considered a community of three people, Alice, who has stored and indexed her publications on her desktop, and Bob and Chris from whom she received some of them. We have created additional annotations referring to the affiliation of some authors as well as regarding the source of the documents. We assumed 10 documents received as email attachments from Chris and 21 received from Bob. Publications are related to each other through citation, and are linked to persons by authorship or source relations. Paths are materialized up to literals, but not including other documents.

In our first experiment we searched for a document about *Web Communities*. If we search with this phrase only in the full-text, we get as hits the documents

119, 61 and 41 in this order and with similar rankings. If we add the title restrictions to the query we get 61, 119 and 14 with relevant differences in the ranking values. If we specify that the metadata query is mandatory, with the *MUST* operator, we only get document 61, 'Self Organization and Identification of Web Communities'.

In the second experiment we search for the phrase "semantic web" in the text field and get 24 documents. If we want to get documents written by persons affiliated to MIT we can narrow the search area by adding the keyword "MIT" to the text query. This approach returns 13 documents. If we add the metadata query '*/creator/affiliatedTo* `MIT.`' or just '*/affiliatedTo* `MIT.`', we get only two documents.

In the third experiment we search for documents received from "Chris Conti" and we got 10 hits, then we searched for documents received from "Bob Doe" and we got 21 hits. When we did a cross search, for documents received from "Chris Doe" we obtained no hits, which is exactly what we wanted to achieve.

In the first experiment, explicit metadata search has a big impact on ranking. Without the metadata restriction, Lucene TF·IDF scores were between 0.04 and 0.07. After adding the metadata phrase to the query, the first document (the one containing the keywords in the title) had a ranking of 0.1471. The terms "web" and "communities" appear less in the text field than in the metadata field. Together with the fact, that the metadata field is much smaller than the full-text field, the ranking considers matches in the metadata field much more important than in the full-text field.

5 Conclusions & Current Work

This paper has presented an integrated approach to indexing and querying full-text and metadata information which is both intuitive for the user and efficient to implement. It achieves this by building on top of a common full-text search engine and by only slightly modifying the usual Google-like search interface to allow the specification of RDF fragments as part of the query. Additionally, our approach provides inexact matching of user queries and ranking of results, evaluating metadata matches the same way as full-text matches, with the ability to return results even for incomplete and over-constrained metadata queries.

References

1. S. Amer-Yahia, P. Case, T. Rölleke, J. Shanmugasundaram, and G. Weikum. Report on the DB/IR Panel at SIGMOD 2005. In *SIGMOD*, June 2005.
2. D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML Documents via XML fragments. In *SIGIR*, New York, USA, 2003.
3. S. Ghita W. Nejdl P. Chirita, R. Gavriloaie and R. Paiu. Activity Based Metadata for Semantic Desktop Search. In *ESWC*, Greece, May 2005.
4. V. Sinha and D. R. Karger. Magnet: Supporting Navigation in Semistructured Data Environments. In *SIGMOD*, June 2005.