Result Ranking for Structured Queries against XML Documents

Torsten Schlieder*
Freie Universität Berlin
schlied@inf.fu-berlin.de

Holger Meuss
Ludwig-Maximilians-Universität München
meuss@cis.uni-muenchen.de

Abstract

XML allows to represent both content and structure of documents. Querying XML data therefore requires a combination of a formal query language with the concept of relevance used in Information Retrieval. In this paper we present such a combination: First, we review Tree Matching as a simple and elegant means to formulate queries without knowing the exact structure of the data. Second, we propose a dynamic document concept by deciding on the document boundaries at query time. Third, we marry structured queries with term-based ranking by extending the term concept to structural terms which include substructures of queries and documents. We show how the notions of term frequency and inverse document frequency can be adopted to dynamically defined documents and structural terms. We introduce an efficient technique to calculate both term frequencies and inverse document frequencies at query time. By adjusting parameters of the retrieval process we are able to model two contrary approaches: the classical Vector Space Model, and the original Tree Matching approach.

1 Introduction

XML gains growing importance in the field of Digital Libraries, bridging the gap between the Information Retrieval (IR) community and the field of database research. It offers a uniform and standardized way to represent and exchange both documents in the sense of IR and data stored in databases. So far, both communities faced the new challenges in their own ways: Database research migrated query languages for semistructured data to XML [ABS00]. But none of the query languages for semistructured data is appropriate for information discovery in digital libraries, since

- the user must *know* the data structure in order to formulate queries, and
- no query language supports *result ranking* according to the user's information need.

Traditional IR techniques, on the other hand, are based on flat text models [BYRN99]. They cannot be used for XML without serious restrictions, since

- the *document structure* is ignored which dramatically lowers the query precision, and
- the *document concept* is static and typically bound to physical documents.

To understand the problems arising from predefined document boundaries please consider XML documents describing books. Some documents may contain information about a single book. Others may collect data about a whole library consisting of thousands of books. Obviously, it is no good idea to retrieve a whole library when the user is only interested in a book. Sometimes the user may only be interested in a chapter or section. This problem can only be solved with a dynamic document concept – but a dynamic document concept does not allow a precalculation of basic values needed for ranking like *term frequencies* and the *inverse document frequencies*.

*This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316).

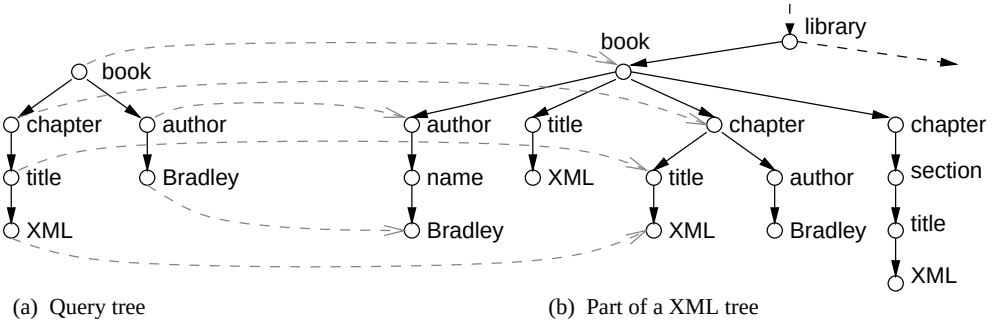

Figure 1: Unordered inclusion of a query tree in an XML tree.

In this paper we tackle each of the above problems. In Section 2 we review Tree Matching as a simple and elegant technique to query XML documents. Tree Matching allows to formulate structured queries with only *partial* knowledge of the document structure. In Section 3 we introduce a solution to the document-boundary problem by deciding on the document type at query time. Section 4 proposes an approach to combine structured queries with term-based *ranking*. We extend the term concept to *structural terms* which include substructures of queries and documents. This makes it possible to assign positive relevance values to XML documents that only partially match the query. In Section 5 we show how the ranking mechanism of the classical Vector Space Model can be extended to dynamically defined documents and structural terms. In Section 6 we demonstrate that our approach can simulate both the Vector Space Model and the Tree Matching formalism by adjusting weights of the query vector. In Section 7 we introduce an efficient technique to compute all necessary term frequencies and inverse document frequencies at query time. In Section 8 we present first experiences with the implementation of our approach. Finally, we review related work in Section 9, and give a conclusion in Section 10.

2 Tree Matching

Kilpeläinen introduced in [Kil92] several variations of Tree Matching techniques as simple and intuitive means to answer queries over trees. We decided to use a slightly modified version [Meu00] of *unordered tree inclusion* as the most flexible and powerful variant[1]. Figure 1 shows an unordered tree inclusion of a query tree in an XML tree. A match is defined using a mapping from the query nodes to the nodes of the XML tree that preserves label names and the ancestor-descendant relation. Note, that the order of siblings is not important, and that some nodes may be skipped in order to include the query tree in the XML tree (node name in the example). We call the image of the query root a *match*. The subtree rooted at a match is returned as a *result*.

3 Logical Documents

We model a collection of *physical* XML documents as labeled trees connected by a single root with a unique label. Currently, we ignore ID-references and hyperlinks. To simplify our model, we only use a single node type for XML elements, attributes, and text data using normalization techniques described, for example, in [SN00].

The definition of Tree Matching results leads to a dynamic document concept: Every subtree of the XML tree is considered to be a *logical document*. We assign types to queries and documents: The *type* of a query and a logical document, respectively, is determined by its root label. Thus, a book query only selects book documents.

[1] Note that our formalism can also be combined with the nine other variants of Tree Matching.

4 Structural Terms

In IR, the common notion of terms is restricted to unstructured terms (in most cases simply words). We extend the definition of terms to substructures of the query tree and the XML tree in order to achieve three objectives: First, we need a notion to express that a query may have partial matches only. In our model, a query has a partial match if at least one query substructure occurs in a logical document. Second, we want to measure how good a query fits to the data by considering how many query substructures have matches in a logical document. Third, we want to reflect the observation that not only flat terms but also substructures have a distribution in and among logical documents.

Due to the lack of space we only give an informal definition: Every labeled tree (with a given alphabet of labels) is called *structural term*. A structural term has an *occurrence* in a query if it is a subtree of the query tree. The definition of an occurrence in a document slightly differs: A structural term has an *occurrence* in a document if the term has a *match* in the XML subtree representing the document.

Example 4.1 *Six structural terms occur in the query in Figure 1: The two "atomic" subtrees consisting only of the nodes labeled* XML *and* Bradley*, respectively. Next, the three subtrees rooted at the inner nodes with labels* title*,* chapter*, and* author*, respectively. Finally, the query tree itself is a structural term. The XML document in Figure 1 has many more structural term occurrences, e.g. the four structural terms* book[author]*,* Bradley*,* author[name[Bradley]]*,* book[Bradley,title[XML]]*.*

Structural terms behave like ordinary terms: They have a number of occurrences in a logical document and a distribution across all logical documents of a given type. Therefore, we can adopt the standard definitions of *term frequency* (tf) and *inverse document frequency* (idf) for structural terms: The tf of a term t_k is the number of its occurrences in the logical document d_i normalized by the frequency of the most frequent term in document d_i. The idf of t_k reflects the ratio between the number of all logical documents and the number of documents containing term t_k.

5 Result Ranking

The Vector Space Model (VSM) [SM83] is one of the most popular models used in IR. It is based on the comparison of the query term vector with the document term vectors. Each term has a certain weight which reflects its descriptiveness with respect to the query or document.

We extend this model to term vectors consisting of *structural terms*. That is, each component of the query and document vectors contains the weight of a structural term. Since we use the standard definitions of tf and idf (adopted to our dynamic document concept), we are able to employ the formula $w_{ik} = \mathsf{tf}_{ik} \cdot \mathsf{idf}_k$ to determine the term weight of term t_k in document d_i.

The same analogy holds for the comparison function between a query vector and a document vector, assigning the document a relevance value with respect to the query. We are free to use any function developed for the VSM, e.g., the Cosine, Dice or Jaccard function. We use the scores computed by one of these functions to generate the ranked result list.

However, there is a difference to the classical VSM. Our terms are *structurally dependent*, i.e., some terms are subtrees of others both in the query and the documents. These dependencies are, in contrast to the dependencies of the VSM, necessary for our model: Assigning weights to structural terms that contain each other allows us to improve the *precision* without lowering the *recall*. Please see Figure 1. Every document that contains the query term author[Bradley] does also contain the term Bradley. But documents that contain the "larger" term author[Bradley] are preferred since their score is a function of the weights of both terms. Consequently, logical documents that match to the whole query get an extra reward reflected by the weight assigned to the structural term representing the query.

6 Simulating Classical Models

Our approach generalizes the classical VSM for flat text and the original Tree Matching approach. We simulate the classical VSM by masking all "complex" structural terms, i.e, we assign 0-values

to all positions of the query vector that do not correspond to the leaf nodes of the query. Therefore, only "flat" terms of the documents are incorporated in the computation of the similarity score. Consequently, the scores computed by the classical VSM and by our model are exactly the same.

We simulate the Tree Matching approach in a similar way: Only the structural term representing the whole query gets the weight 1; all other components of the query vector are set to 0. With this technique, only full matches of the query achieve a score greater than 0. Non-matching documents as well as documents with partial matches all get the score 0.

7 Implementation

The typical implementation of the VSM consists of a lexically sorted list of all terms occurring in the document collection. Each term is annotated with the idf and a list of occurrences and term frequencies. We cannot use this simple technique for two reasons: First, the set of structural terms that have matches in the data tree is exponentially larger than the set of document terms in the flat text model. Second, the scope of the term frequencies are *logical* documents which are defined at query time. The number of potential logical documents is equal to the number of inner nodes of the data tree. If we computed all term frequencies at indexing time, the posting of a term would consist of references to all data subtrees containing the term. Notice also, that a query selects only documents of a certain type. Hence, each term must refer to a list of idfs – one idf for each document type. We therefore compute the tf and idf values while executing the query. Fortunately, we can restrict our computations to structural terms occurring in the query, and to the document type determined by the label of the query root.

Our notion of tree embedding requires that every parent-child pair of a query term must be mapped to two data nodes for which the ancestor-descendant relationship holds. To construct all ancestor-descendant pairs for a given pair of labels we adopt an indexing technique called *partial index* introduced in [Nav95]. The partial index consists of an inverted list of all index terms, and a structural index. Each entry of the inverted list indicates all text positions in which the term occurs. The structural index consists of all node types (i.e., different node labels). Each type refers to all its occurrences using two sorted lists: The first one stores the start positions of text segments covered by the node. The second list indicates the final positions of the segments.

Our approach differs from [Nav95] since we (1) model a document collection as a single tree, (2) allow recursive labeling, and (3) allow not only text but also attribute values and empty elements to be leaf nodes of the XML tree. However, the most important extension is the computation of all required term frequencies and inverse document frequencies at *query time*. We count the number of occurrences of a structural term in a logical document, and the number of logical documents containing the structural term. We add a third list to each node type of the structural index. Each list entry stores the frequency of the most frequent index term occurring in the tree rooted at the corresponding data node. We use these values to normalize the computed term frequencies without additional time complexity.

In the case of non-recursive trees the time complexity of our algorithm for the ancestor-descendant problem is bound by $O(s)$. The parameter s denotes the selectivity, i.e., the maximal number of nodes having the same label. If the tree contains recursive labels, parts of the inner relation may be processed several times. Since the number of nodes carrying the same label along a path cannot exceed the depth d of the tree, the complexity is bound by $O(s \cdot d)$. To compute the tf and idf for all documents of a certain type we need at most $2q-3$ path joins, where q is the number of query nodes. Hence, the overall complexity of our algorithm is $O(q \cdot s \cdot d)$.

8 Practical Experiences

In this section we present first experiences with the implementation of our model. Since there is currently no large test collection of XML files, we decided to use a small set of legal documents provided by the juris GmbH, Saarbrücken. This collection consists of 22 documents containing laws, regulations, and directives of the German federal state Hessen. We assumed several sample information needs, formulated different queries for each, varied the query weights, and studied the rankings generated

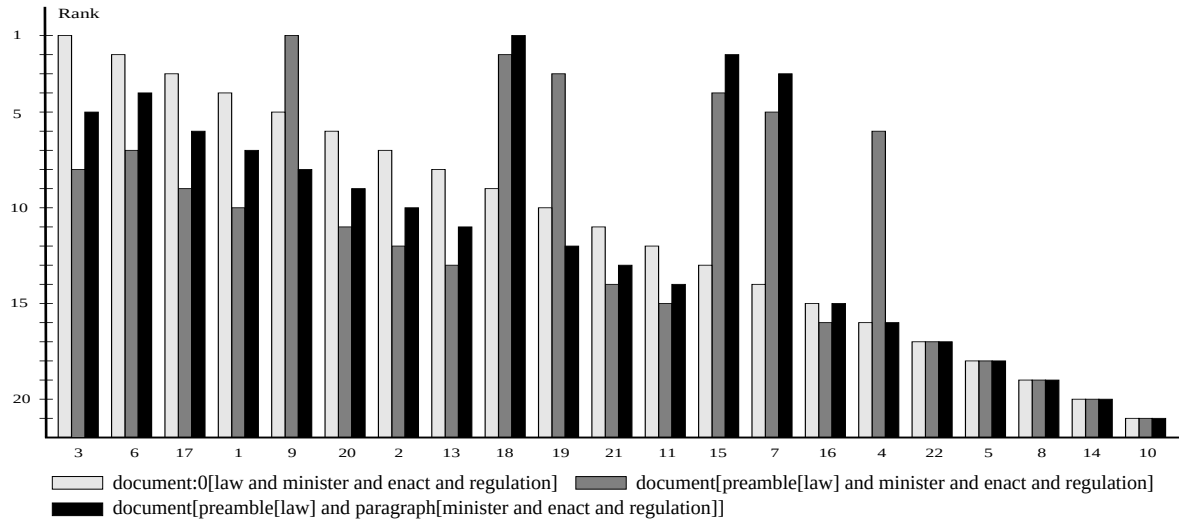

Figure 2: Ranking the example collection with different queries

by the system. We learned that structured queries do not necessarily lead to a better ranking if the keywords are semantically related, i.e., tend to occur together in a data subtree. However, we observed a strong improvement (1) if the query specifies that certain terms should be in the metadata of the logical documents (e.g., title, preamble), or (2) if the query terms are not closely semantically related.

We illustrate the improvement of the retrieval precision using the sample information need "Get all laws that describe how ministers enact regulations." Our first query (depicted in Figure 2 with light grey bars) is unstructured, containing only the keywords law, minister, enact, and regulation. We applied the weight 0 to the whole query in order to simulate the flat Vector Space Model (see Section 6). The second query (dark grey) additionally requires that law appears in the preamble of the document. The query also prefers full matches over partial matches by using the implicit weight 1 for the structural term representing the full query. The third query (black) extends the second one and expects the keywords minister, enact, and regulation to occur in the same paragraph.

Figure 2 shows the the rankings generated by the three queries. The x-axis indicates the logical document numbers; the y-axis shows the relative position of the document in the ranking. We sorted the documents according to the ranking generated for the first query.

The figure points out that adding structure to a query does not lower the recall but strongly improves the query precision. The shift from the flat query to the second one prefers documents that are in fact laws – which is specified in the document preamble. That is, the documents 9,18,19,15,7,4 now take the first positions in the ranking. Note, that we are also able to weaken this effect by applying a smaller weight to the query node preamble. With this kind of query extensions we are able to involve *metadata* in the text retrieval process – but our query does not fail if the metadata does not contain the specified terms.

By constraining the remaining keywords to appear in the same paragraph we again change the ranking order. Now, documents of type law that contain all three keywords in a *single* paragraph take the first positions in the ranking (documents 18,15,7). And indeed, the documents 18, 15, and 7 contain the information satisfying the original information need – while other documents may also contain all keywords but not in the same context. In fact, the inner query node paragraph works similar to the near operator used in some retrieval engines but does not rely on positional distances but on *semantic closeness* specified by the document creator.

Another important observation is the *stability* of the rankings. Documents that have many matches of query subtrees are preferred with respect to the ranking generated by the flat query – but documents that have only partial matches preserve their relative positions to each other. This can be seen in the diagram in Figure 2 by sequences of documents $(6, 17, 1, 9, 20, 2, 13)$ or $(22, 5, 8, 14, 10)$, that mainly keep their descending ranking order for all three queries.

9 Related Work

There have been few research results concerning the ranking of structured queries against XML documents. In [BYRN99], several approaches that combine content and structure are compared. However, none of the models considers relevance ranking. XXL [TW00] and XIRQL [FG00] introduce formalisms loosely based on XML-QL and XQL, respectively, that incorporate the notion of relevance. In both cases the relevance of a document to a structured query is determined by combining the weights of the terms in the query leaves according to the query structure. In contrast to our approach, neither XXL nor XIRQL can treat partial structural matches, where only a part of the query structure matches a document. In addition, the question of how the term weights in the documents can be adjusted to the dynamic document notion is not discussed. The static assignment of weights to document terms at indexing time used in XXL and XIRQL cannot reflect the weight of a term in the varying notions of a logical document determined at query time. The formalism approXQL, as proposed in [SN00], is based on Tree Matching and follows an orthogonal approach to XXL, XIRQL, and our model: Embeddings that skip few document nodes are preferred and result in a high relevancy value for the respective document.

10 Conclusion

In this paper we proposed an approach to combine structured queries against XML documents with result ranking. With our model we are able to improve the precision of the retrieval process without lowering the recall. Moreover, our approach generalizes both the classical VSM and the Tree Matching approach by simply adjusting weights of the query vector. We also presented an implementation technique that allows us to efficiently compute all necessary term frequencies and inverse document frequencies at query time. First experiments with our prototype have shown that incorporating structure into the VSM improves the retrieval precision especially for queries with keywords that appear in different contexts. We plan to do further experiments to study the behavior of the model with different collections and queries.

References

[ABS00] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, San Francisco, 2000.

[BYRN99] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley Longman, 1999.

[FG00] N. Fuhr and K. Großjohann. XIRQL: An extension of XQL for information retrieval. In *ACM SIGIR Workshop On XML and Information Retrieval*, Athens, Greece, July 2000.

[Kil92] P. Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases.* PhD thesis, University of Helsinki, Finland, November 1992.

[Meu00] H. Meuss. *Logical Tree Matching with Complete Answer Aggregates for Retrieving Structured Documents.* PhD thesis, Dept. of Computer Science, University of Munich, 2000.

[Nav95] G. Navarro. A language for queries on structure and contents of textual databases. Master's thesis, Department of Computer Science, University of Chile, April 1995.

[SM83] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Tokio, 1983.

[SN00] T. Schlieder and F. Naumann. Approximate tree embedding for querying XML data. In *ACM SIGIR Workshop On XML and Information Retrieval*, Athens, Greece, July 2000.

[TW00] A. Theobald and G. Weikum. Adding relevance to XML. In *WebDB'2000 (Informal Proceedings)*, 2000.