A Probabilistic Retrieval Model for Semistructured Data

Jinyoung Kim, Xiaobing Xue and W. Bruce Croft

Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts Amherst
{jykim,xuexb,croft}@cs.umass.edu

Abstract. Retrieving semistructured (XML) data typically requires either a structured query such as XPath, or a keyword query that does not take structure into account. In this paper, we infer structural information automatically from keyword queries and incorporate this into a retrieval model. More specifically, we propose the concept of a mapping probability, which maps each query word into a related field (or XML element). This mapping probability is used as a weight to combine the language models estimated from each field. Experiments on two test collections show that our retrieval model based on mapping probabilities outperforms baseline techniques significantly.

1 Introduction

Despite the huge amount of unstructured text data on the web, structured databases support most web services as a back-end, and XML is becoming increasingly popular as the format for semistructured data. Therefore, effective retrieval of structured or semistructured data continues to be a significant research issue for many applications.

In this paper, we focus on the retrieval of semistructured data in the form of XML documents where the fields (or XML elements) have primarily textual content. This type of data often has considerable overlap in the content and use of elements. Most user queries for such data will target one or more of these elements, which leads to search interfaces where users enter query specifications into multiple fields.

To motivate the approach described in this paper, we start with an example, where a user wants to find a romantic movie featuring Meg Ryan. In the IMDB[1] website, searching with a simple query 'meg ryan romance' does not produce any relevant movies in the top ranked list. If the "power" search interface is used, the user can specify 'meg ryan' in the cast field and 'romance' in the genre field, which finds a number of movies such as 'Sleepless in Seattle'.

From this example, it is evident that providing structural information for each part of a query can be more effective. This raises the question of whether

[1] Internet Movie Database: http://www.imdb.com

we should always ask users to provide this information for each query. Considering that many users are generally only willing to express the simplest forms of queries, this may be an excessive burden. This is supported by the observation[2] that “advanced” search functions involving field-based queries are not used by most users. Explicitly issuing structured queries such as SQL or XPath is also beyond the capability of most users since it requires the knowledge of the schema as well as the query language.

The focus of the work described in this paper is to shift the burden of specifying effective queries for semistructured data from the user to the system. Our approach is to have the system “guess” the structural information that is implicit in keyword queries. In the next section, we provide a general description of this approach.

1.1 Inferring the User’s Intent

We begin with the observation that a query has an implicit mapping of each query term into XML element(s), even though users may not want to fill out complex forms or formulate structured queries. Imagine a user is trying to find a movie description XML document composed of elements like title, genre, cast, and so on. If she issues the query ‘meg ryan romance’, it would be desirable if the system can infer the mapping between these query terms and the corresponding XML elements (*cast* and *genre*) without any additional specification.

The distribution of words in the elements in the database provides clues for this inference process. In our example query for the IMDB database, the query term ‘romance’ will be found more frequently in the *genre* elements, while the terms ‘meg’ and ‘ryan’ occur mostly in the *cast* or *team* elements. An important assumption of our work is that each XML element will have a distinctive distribution of terms, which, in the case of the IMDB database, is reasonable considering that different XML elements describe different aspects of the movie.

Given a mapping between query terms and XML elements, how can we use this information for retrieval? One way is to convert keyword queries into structured queries. In the example above, the resulting XPath query would be `//movie[actor=‘meg ryan’ and genre=‘romance’]`.

While this is a reasonable approach, we expect that there will be an inherent loss of information in this conversion process, since it is likely that only a few of the top ranked elements will be used in the resulting structured query. In this case, if some query terms were mapped into elements not intended by the user, we cannot expect to get the right result. Extending the movie example, if a query word is inferred to be an actor’s name by the system yet actually was a director’s name, no relevant movie will be returned. A structured query also in general returns a set, although in most cases a ranked list will be more desirable, especially if there are too many items in the result set or the XML document contains full-text elements.

[2] http://www.useit.com/alertbox/20010513.html

Accordingly, we take an IR approach and regard the mapping relationship as the strength of evidence each element provides for each query term. In other words, if an element (e.g., *genre*) is given the highest mapping probability for a given query term (e.g., 'romance'), then the occurrence of the query term in that element is assigned more weight than any other elements, reflecting the inference that the user may have meant the type *genre* for the query term 'romance'. This retrieval model can exploit the mapping without the loss of information in that every element can contribute a score. In addition, the result is naturally a ranked list of documents.

In summary, we try to infer the user's query intent on a per-term, per-element basis, finding which document element each query term may be associated with. Then we incorporate the resulting mapping into the traditional language modeling approach to IR [11] to combine element-level scores into a document score.

The rest of this paper is organized as follows. We introduce related work in Section 2, followed by our mapping probability estimation technique and the corresponding retrieval model in Section 3. Then we provide experimental verification of this approach in Section 4. Lastly, we conclude with a discussion of potential future work.

2 Related Work

Related work can be mostly found in investigations of the XML retrieval task, which has been addressed from both the IR and database perspectives. This work is also similar to the research on keyword search over relational databases, since the task is basically ranked retrieval of structured data using keyword queries.

The INEX initiative is a major study of XML retrieval [2]. The INEX adhoc track addresses the task of retrieving XML data with explicit document structure, such as section and title, and has used test data consisting of scientific papers or Wikipedia articles. Our focus instead is on XML data where each element type has more specific semantics and, consequently, a distinctive term distribution. This is consistent with the view that the database schema represents the important semantics of the data rather than the structural layout. Also, whereas INEX addresses the issues of finding the right granularity for XML retrieval and the corresponding evaluation metrics, we limit our task to retrieving entire documents (or XML records).

A recent paper from INEX [8] suggested an extension of the classic probabilistic retrieval model where each term score is weighted by tag (element type) score. A tag score for each term is estimated based on the probability that the element judged relevant contains the term. This method is similar to the one presented in this paper in that it weights element-level scores using tag scores, but it does require element-level relevance judgments. Evaluations of this method were inconclusive. The BM25f model [13] also has some similarity to our approach.

Other recent work [10] showed that a keyword query can be refined into a structured query by mapping each query term into a set of structural fragments and transforming these fragments into the XPath query that represents the original information need most appropriately. While the initial mapping step of this method bears some resemblance to our mapping probability estimation and can be done using collection statistics, the subsequent conversion into a structured query has the potential problem of missing information that we mentioned in Section 1.1. Calado et al [3] describe a method of ranking candidate structured queries that is similar to the one described in this paper, although it was applied and evaluated differently.

The database community has also studied XML retrieval with keyword queries. The concept of Lowest Common Ancestor (LCA) [5] has been proposed to answer keyword queries, where the LCA corresponds to the lowest-level XML element which contains all query words in its descendant elements. Besides XML retrieval with keyword queries, there has been work about keyword search in relational databases, which includes DBXplorer [1], DISCOVER [6]. For these systems, the answers to the keyword query are the tuple trees joined from multiple tables containing query words.

3 A Probabilistic Retrieval Model for Semistructured Data

In this section, we introduce the probabilistic retrieval model for semistructured data. We start by discussing the hierarchical language model, which is the basis of our method. Then we explain how we can infer mapping probabilities from collection statistics and incorporate them into the retrieval model.

The following notation will be used throughout this paper. We will assume that a query $Q = (q_1, q_2, ..., q_m)$ is composed of m words and the collection C built on a single XML Schema $E = (E_1, E_2, ..., E_n)$ is composed of n element types. Each document d in the collection is composed of elements $(e_1, e_2, ..., e_n)$, where each element is marked using lowercase letters to distinguish it from the corresponding element type in the schema.

Note our simplifying assumptions that the document as a whole is the unit of retrieval and every document has only one level of elements. The model can be extended to relax these assumptions, such as dealing with a hierarchy of elements.

3.1 Hierarchical Language Model

Ogilvie and Callan [9] suggested the hierarchical language model (HLM), where they proposed a language modeling approach to IR adapted for XML component retrieval, by smoothing with parent/child elements and differential weighting for each element type.

Based on our simple one-level schema E, a document score in the HLM approach is formed by taking a weighted average of element-level scores as expressed in the following:

$$P(Q|d) = \prod_{i=1}^{m} \sum_{j=1}^{n} \mu_j P_{QL}(q_i|e_j) \tag{1}$$

$$P_{QL}(q_i|e_j) = (1-\lambda)P(q_i|e_j) + \lambda P(q_i|E_j) \tag{2}$$

$P_{QL}(q_i|e_j)$ is the query-likelihood score of element e_j after appropriate smoothing with the background language model built using all instances of E_j. Callan and Ogilvie also suggest guidelines for determining n weight parameters μ_j for each element, such as setting them proportional to the length of text content or the importance for retrieval.

However, the weights in HLM are fixed regardless of the query once they are set. This can be too restrictive if each element provides a different strength of evidence for different query terms. In view of this, our work can be seen as an extension of HLM where we assign different element-level weights that are estimated for each query term.

3.2 Mapping Probabilities

As briefly discussed in the introduction, we can infer the mapping between each query term and XML element based on collection statistics. More formally, using Bayes' theorem, we can estimate the posterior probability $P_M(E_j|w)$ that a given query term w is mapped into XML element E_j by combining the prior probability $P_M(E_j)$ and the probability of a term occurring in a given element type $P_M(w|E_j)$.

$$P_M(E_j|w) = \frac{P_M(w|E_j)P_M(E_j)}{P(w)} = \frac{P_M(w|E_j)P_M(E_j)}{\sum_{E_k \in E} P_M(w|E_k)P_M(E_k)} \tag{3}$$

$P_M(w|E_j)$ is calculated by dividing the number of occurrences for term w by total term counts in the element E_j across the whole collection. In other words, $P_M(w|E_j)$ is the probability of generating word w from a virtual document created by combining all E_j elements in the collection. Also, $P_M(E_j)$ denotes the prior probability of element E_j being mapped into any query term before observing collection statistics.

Following this estimation procedure, the mapping probability can be viewed as a normalized query-likelihood score for each element type given a query. It can also be interpreted as an effort to capture the 'significance' of a query term for each element type. For instance, a query term 'romance' can be found in both *genre* and *plot* elements of a movie XML document, yet 'romance' is more significant in *genre* than in *plot* element and therefore it is more likely to be the element the user intended.

Another point worth remarking is that it is relatively cheap to calculate this mapping probability since it is based on collection statistics, which is already available in the search engine index. Therefore, no additional indexing is required for mapping probability calculation. Also, this can be done before a user issues a query, thereby having virtually no impact on the perceived speed of retrieval.

3.3 Incorporating Mapping Probabilities into the Retrieval Model

With the mapping probabilities estimated as described above, the probabilistic retrieval model for semistructured data (PRMS) can use them as weights for combining the score from each element into a document score, as follows:

$$P(Q|d) = \prod_{i=1}^{m} \sum_{j=1}^{n} P_M(E_j|q_i) P_{QL}(q_i|e_j) \tag{4}$$

Here, the mapping probability $P_M(E_j|w)$ is calculated as described in Section 3.2 and the element-level query-likelihood score $P_{QL}(w|e_j)$ is estimated in the same way as in the HLM approach.

The rationale behind this weighting is that the mapping probability is the result of the inference procedure to decide which element the user may have meant for a given query term. For instance, for the query term 'romance' , this model assigns higher weight when it is found in *genre* element as we assume that the user is more likely to have meant a type of movie rather than a word found in *plot*.

One may imagine a case where the user meant 'meg ryan' to be words in the *title* and 'romance' to be in the *plot*. Given that our goal is to make the best guess with the minimal information supplied by user, however, the PRMS will not rank movies that match this interpretation as highly as the more common meaning. Movies that do match this interpretation will, however, appear in the ranking rather than being rejected outright which would be the case if we were generating structured queries. The experimental results based on collections and queries taken from the actual web services supports the claim that the common interpretation is usually correct.

4 The Experiments

In this section we describe experiments for verifying the effectiveness of the suggested model. Since our retrieval model returns a ranked list of documents given a keyword query, we follow standard IR experimental methodology [4], measuring retrieval effectiveness using a set of queries and getting overall performance score by averaging query-level scores.

We prepared two XML collections, IMDB and Monster. As will be explained in a greater detail in the following, IMDB is viewed as a "clean" collection in the sense that data has been entered into the correct fields. We expect this type of database to be very suitable for our method, whereas the Monster data has much more "noise" and is consequently more challenging. For both collections, each word is stemmed during indexing using the Krovetz stemmer [7]. Stopwords were not removed.

For the retrieval experiments, we used Indri [14], a state-of-the-art search engine based on the language modeling approach to IR [11]. We chose Indri for several reasons. First, it supports indexing of XML documents by their elements, which means that each XML element is considered as a small document for which a relevance score can be calculated. Second, it supports combining scores from each element with arbitrary weights based on an inference network, which is a crucial aspect of our retrieval model. In a sense, our retrieval model can be understood as transforming a given keyword query into a semistructured Indri query expression where different weight is given to each element for each query term.

In addition to the search engine, since we needed a mapping probability for each query term, we created a simple program that calculates mapping probabilities based on the collection statistics available in the Indri index. The program simply reads term occurrences from the index corresponding to each XML element and computes the mapping probability based on Equation 3, which was later used as a weight for Indri queries.

For a comparison of retrieval effectiveness, our baselines were the standard document query-likelihood model (DQL) and a variant of the hierarchical language model (HLM) which assigns a fixed weight for each element according to its presumed importance.

Our experiments require some parameters to be tuned in advance. We had to determine smoothing parameters [15] and element-level weights for HLM. For each collection, we set aside 10 training queries to find the best-performing parameters and used these parameters for test queries. Since the HLM parameters required training for each XML element, we adopted a iterative line search based on the Golden Section Search [12] algorithm.

4.1 Experiments with IMDB Data

We did an initial experiment with the IMDB[3] dataset, which consists of 437,281 “documents” or XML records. Each document corresponds to a movie and was constructed from text data[4]. The element types were ‘title’, ‘year’, ‘releasedata’, ‘language’, ‘genre’, ‘country’, ‘location’, ‘colorinfo’, ‘cast’, ‘team’ and ‘plot’. As seen in the earlier example, document content consists mostly of keywords, with an exception of the *plot* element. Since each element has little overlap in word distribution, it was expected that the mapping probabilities could be estimated with high accuracy.

We prepared 40 queries, assuming the situation where a user wants to find a movie with partial information spanning over many elements. One may remember some words in the title, the name of an actor, or the year it was released and so on. These are all combined into a keyword query, as summarized in Table 1. Since most queries were designed to find a single movie, we could find relevant documents by manual search.

We also present some examples of estimated mapping probabilities in Table 2. As you can see, most query terms appear to be mapped into the expected

[3] An XML collection of movies from the Internet Movie Database: http://www.imdb.com

[4] Available in http://www.imdb.com/interfaces#plain

Table 1. Example queries for IMDB collection.

#	Query	Description
1	Love Letter Iwai	Movie with title 'Love Letter', directed by Iwai Shynji
2	Ziyi Zhang hidden tiger	'Crouching Tiger Hidden Dragon' featuring Ziyi Zhang
3	Meg Ryan war	A war movie featuring Meg Ryan
4	Redemption crime	'The Shawshank Redemption', a crime movie
5	Brokeback Ang Lee	'The Brokeback Mountain' directed by Ang Lee

elements. Given these mapping probabilities, we automatically formulated example query 3 into the Indri query shown in Figure 1, where the score from each element is combined for each query term. In this query, `#wsum` is the weighted sum operator which supports the weighted combination of evidence, `#combine` is the top-level combination operator, and the '`.`' operator specifies an element of the document.

Table 2. Mapping probability examples from IMDB collection. For each query-term, only 3 elements are used for this example.

Query 3			
meg	cast:0.407	team:0.382	title:0.187
ryan	cast:0.601	team:0.381	title:0.017
war	genre:0.927	title:0.070	location:0.002
Query 4			
redemption	title:0.983	location:0.017	year:0.000
crime	genre:0.990	title:0.010	location:0.000

```
#combine ( #wsum(0.407 meg.(cast) 0.382 meg.(team) 0.187 meg.(title))
           #wsum(0.601 ryan.(cast) 0.381 ryan.(team) 0.017 ryan.(title))
           #wsum(0.927 war.(genre) 0.070 war.(title) 0.002 war.(location)))
```

Fig. 1. Indri query example from IMDB collection.

Out of a total of 134 query terms, 91 (68%) query terms were mapped into the correct element with the highest probability. Including elements with the second highest mapping probability, 113 (84%) were correct. As an example of an incorrect mapping, in the query 'star wars', the term 'wars' was stemmed into 'war' and mapped into a *genre* element rather than a *title* element, which would have been right in this case.

We used Mean Average Precision (MAP) and Reciprocal Rank (RecipRank) as the performance metrics. MAP is the mean of query-wise overall precision averaged for each relevant document, which is the most widely used performance measure. When most queries have only one relevant document, as in this experiment, Reciprocal Rank is a common measure. This measure is simply $1/r$ where r is the rank of the first relevant document found.

Table 3 shows the overall retrieval effectiveness for the test queries on the IMDB collection. It is clear that the proposed method performs significantly better (p-value < 0.01 with two-tailed t-test) than both baselines. Another observation is that HLM performed worse than DQL. This seems reasonable given that each query term was targeted for a different element and HLM assigns predetermined weights for each element, which could cause worse performance than not assigning weights at all (which is the case for DQL).

We notice that the performance of the HLM model is quite low here. A further explanation is that, in the HLM, the importance of different fields is the same for different query words. This assumption is invalid for our query set, however, where each query word is aimed at different fields. Consider the query "gladiator action scott" for example, which should find the movie "Gladiator". Here, the word "gladiator" is targeted for the "title" field, whereas the word "action" is targeted for the "genre" field and so on. Table 4 shows the weights used by PRMS and HLM, where it is clear that high performance of PRMS comes from the flexible weights of the fields, while a set of fixed field weights (based on a training set of queries) accounts for low performance of HLM.

Table 3. Retrieval performance for IMDB collection. Percentages are improvements over the DQL baseline.

Measure	DQL	HLM	PRMS
MAP	0.374	0.344(-8.8%)	0.630(68.4%)
RecipRank	0.405	0.350(-15.7%)	0.635(56.8%)

Table 4. Field weights(for 4 elements) and the retrieval performance for the query "gladiator action scott".

PRMS - RecipRank : 1.0				
gladiator	title:0.634	genre:0.000	plot:0.25	team:0.000
action	title:0.005	genre:0.971	plot:0.01	team:0.000
Scott	title:0.011	genre:0.000	plot:0.020	team:0.440
HLM - RecipRank : 0.01				
all terms	title:0.229	genre:0.108	plot:0.024	team:0.048

4.2 Experiments with Monster Data

Since the IMDB collection does not contain much full text content and the test queries were designed mostly for demonstrating the effectiveness of our method, we performed a second experiment using the Monster[5] job description collection composed of 1,034,795 XML documents.

This was a more realistic setting for many applications since documents were longer, with mostly full-text content. The 60 queries we used were requests for

[5] Collections and queries were licensed from Monster: http://www.monster.com

positions created by real users of the Monster service. For instance, a query 'executive assistant power point excel type schedule' was intended to find an executive assistant with proficiency in Microsoft Office Suite who can handle daily administrative duties.

Each document is composed of elements like 'resumetitle', 'summary', 'desiredjobtitle', 'schoolrecord', 'experiencerecord', 'location' ,'skill' and 'additionalinfo'. Given that the entry of data by type was enforced and that people often put data into the wrong elements, there could be considerable word overlap among the different elements. This made the job of estimating mapping probabilities more challenging. For example, the word 'automotive' was mapped into the *desiredjobtitle* element with the highest probability, but the *resumetitle* and *skill* elements also got high scores.

Table 5. Retrieval performance for Monster collection. Percentages are improvements over the DQL baseline.

Measure	DQL	HLM	PRMS
MAP	0.432	0.254(-70.1%)	0.530(22.7%)
Prec@5	0.51	0.39(-30.8%)	0.61(19.6%)
Prec@10	0.502	0.322(-55.8%)	0.577(14.9%)
Prec@20	0.483	0.26(-86.1%)	0.545(12.8%)

Table 5 shows the retrieval performance of our methods for the Monster collection. The PRMS method still performs significantly better (p-value < 0.01 with two-tailed t-test) than the baseline methods, although the differences are smaller here. HLM is still worse than DQL, reflecting the fact that query terms were targeted into many different XML elements and assigning fixed weights can hurt performance.

It was not the case this time that our method performed better than the baseline for all cases. As an example of where our method was worse, for the query 'toxicology lab supervisor wastewater oakland', the words 'toxicology' and 'lab' were mapped into the wrong element 'experience', while relevant documents contained these words in 'jobtitle'. As seen in this case, mapping query terms into elements not intended by user explains the smaller performance improvement of the PRMS method compared to the IMDB collection.

We also wanted to investigate how performance would change if we used only elements with higher mapping probabilities for each query word, to simulate the loss of information when a given keyword query was converted into a structured query. Comparing the result of retrieval using elements with only the top-k mapping probabilities for the IMDB and Monster collections in Table 6, we could find that performance gets worse as we use only the first few elements with highest mapping probabilities. The degree of degradation was found to be greater in the Monster collection, where mapping probability calculation was more challenging.

Lastly, for the Monster collection, we did an experiment with the prior mapping probability $P_M(E_j)$ as defined in Section 3.3. This prior probability allows us to represent that some elements are more important than others regardless of query terms. Since our analysis for the Monster collection revealed that most queries could be related to *resumetitle* and *desiredjobtitle*, we assigned higher prior weights for these two element types, thereby effectively simulating prior probabilities. The results show that this modification gives a slight boost compared to the original PRMS model, although the margin is not statistically significant.

Table 6. Retrieval performance for IMDB and Monster using top-k elements. k is the number of elements (with highest mapping probabilities) used for retrieval.

IMDB					
Measure	PRMS-top1	PRMS-top2	PRMS-top3	PRMS-top4	PRMS-top5
MAP	0.413	0.572(38.6%)	0.618(49.7%)	0.630(52.7%)	0.630(52.7%)
Monster					
Measure	PRMS-top1	PRMS-top2	PRMS-top3	PRMS-top4	PRMS-top5
MAP	0.260	0.403(54.8%)	0.483(85.2%)	0.497(90.7%)	0.512(96.2%)

Table 7. Retrieval performance for Monster using higher prior weights for title fields. 40% higher weights for PRMS-PW1.4, 80% for PRMS-PW1.8.

Measure	DQL	PRMS	PRMS-PW1.4	PRMS-PW1.8
MAP	0.423	0.534(26.3%)	0.539(27.5%)	0.538(27.2%)
RecipRank	0.640	0.716(11.9%)	0.715(11.7%)	0.715(11.7%)

5 Conclusion

In this paper, we developed a method to infer the implicit mapping between query terms and XML elements, and proposed a novel retrieval model exploiting the resulting mapping probabilities. Our experimental results show that the proposed method, the probabilistic retrieval model incorporating mapping probabilities, improves retrieval effectiveness significantly over baseline methods in realistic settings. We also observed that more performance gain is achieved when mapping probability estimation is more accurate.

Recently we did an additional experiment for the IMDB collection using 1,200 queries targeted for the IMDB movie description webpages, taken from MSN query log data. Again, the result verified our approach, where our method showed significantly better performance over baseline methods. We are also working on further verification of the suggested retrieval model in other settings.

Our approach for structured document retrieval leaves many interesting challenges. Since this method is orthogonal to most existing works for XML retrieval, it can be combined with other techniques to improve the retrieval effectiveness further. The most obvious extension is applying our retrieval model to retrieve elements as opposed to the whole document. We can also straightforwardly extend this term-based mapping probability into a phrase-based one, by finding the likelihood of each element containing a phrase found in the query instead of a term.

6 Acknowledgements

This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF grant #IIS-0707801, and in part by NSF grant #IIS-0711348. Any opinions, findings and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsor.

References

1. S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: enabling keyword search over relational databases. In *SIGMOD Conference*, page 627, 2002.
2. S. Amer-Yahia and M. Lalmas. Xml search: languages, inex and scoring. *SIGMOD Record*, 35(4):16–23, 2006.
3. P. Calado, A. S. da Silva, R. C. Vieira, A. H. F. Laender, and B. A. Ribeiro-Neto. Searching web databases by structuring keyword-based queries. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 26–33, New York, NY, USA, 2002. ACM.
4. C. W. Cleverdon. The significance of the cranfield tests on index languages. In *SIGIR*, pages 3–12, 1991.
5. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD Conference*, pages 16–27, 2003.
6. V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.
7. R. Krovetz. Viewing morphology as an inference process. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development Information Retrieval*, pages 191–203. ACM, ACM, 1993.
8. F. T. M. Géry, C. Largeron. Probabilistic document model integrating xml structure. *Proceedings in INEX*, pages 139–149, 2007.
9. P. Ogilvie and J. Callan. Hierarchical language models for xml component retrieval. In *INEX*, pages 224–237, 2004.
10. D. Petkova, W. B. Croft, and Y. Diao. Refining keyword queries for xml retrieval by combining content and structure. *CIIR Technical Report*, 2008.
11. J. Ponte and W. B. Croft. A language modeling approach to information retrieval. pages 275–281, New York, NY, 1998. ACM, ACM.
12. W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK, 2nd edition, 1992.
13. S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 42–49, New York, NY, USA, 2004. ACM.
14. T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries, 2005. poster presentation.
15. C. Zhai and J. D. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.