A Result Reconstruction Method for Effective XML Fragment Search at INEX 2010

Atsushi Keyaki[1], Kenji Hatano[2], and Jun Miyazaki[3]

[1] Graduate School of Culture and Information Science, Doshisha University
1-3 Tatara-Miyakodani, Kyotanabe, Kyoto 610-0394, Japan
keyaki@ilab.doshisha.ac.jp,
[2] Faculty of Culture and Information Science, Doshisha University
1-3 Tatara-Miyakodani, Kyotanabe, Kyoto 610-0394, Japan
khatano@mail.doshisha.ac.jp,
[3] Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0192, Japan
miyazaki@is.naist.jp,

Abstract. In this paper, we propose an extension scheme of the result reconstruction method that we previously proposed. In the previous method, we aimed to extract more relevant fragments without irrelevant parts. Since the challenge of Ad hoc track in INEX 2010 is to generate "snippets" which return only small size of contents, we extend our method along the concept of INEX 2010. To generate meaningful snippets, we propose a novel way of extracting search results and adjust our previous methods to be suitable one for restricted results in its text size.

Keywords: XML fragment search, result reconstruction method, meaningful snippets

1 Introduction

The extensible markup language[4] (XML) is a markup language for structured documents that has become the de facto format for data exchange. A large number of XML documents are available on the Web. We expect this trend to continue in the future. As such, information retrieval techniques for searching XML documents are very important and required in the field.

An XML fragment is usually defined as a part of a larger XML document. The fragment is identified by the surrounding start and end tags. XML fragments are, therefore, nested and have containment relationships with each other, which can cause overlaps in XML fragments in an XML document. The key goal of XML search is to obtain relevant XML fragments for a query instead of just returning the entire XML documents. Therefore, XML search engines can generate a ranked list composed of a set of relevant XML fragments, while several Web search engines return a set of entire Web documents. By isolating the XML

[4] http://www.w3.org/TR/REC-xml/

fragments, users do not need to identify the relevant XML fragments from the larger XML documents.

INEX Ad hoc track requires a ranked list which contains relevant XML fragments for a query. In addition to this, the Ad hoc track in 2010 requires very restricted results in the text size. This constraint is reasonable because users should want more focused results.

In our previous studies[5][6], we focused on how we identify the relevant fragments without irrelevant parts in each document. Even though these methods could improve search accuracy, they did not consider to return restricted results because they aimed to extract relevant fragments as much as possible. Therefore, we should modify these methods to return suitable results for the Ad hoc track in 2010.

To tackle this challenge, we propose a method for returning ideal search results. We consider the requirements of meaningful snippets while the previous methods focused on how to identify the appropriate granular fragments, because it returns very small sized texts as search results.

2 Result Reconstruction Method

In this section, we show our previous study[6]. In the study, we generate the *refined ranked list*, which is composed of relevant and informative XML fragments. As shown in Figure 1, the overview of the method is as follows:

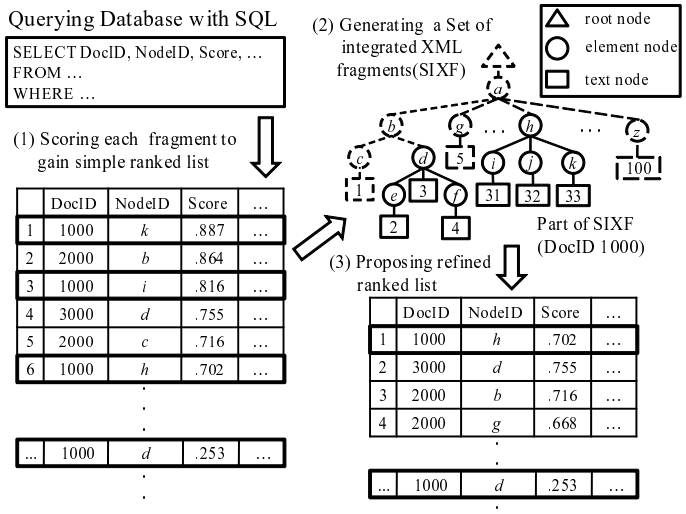

Fig. 1. Overview of the previous study

(1) We first score each XML fragment by using a scoring method to obtain a simple ranked list. We use BM25E[9] to calculate relevance scores of each fragment for a query.
(2) We extract XML fragments from a simple ranked list and generate a set of relevant XML fragments by considering limitations of the XML fragment length and reconstruction of XML fragments.
(3) To present a refined ranked list, we rank the XML fragments extracted from Step (2); we re-rank and remove the XML fragments in the simple ranked list and incorporate them into the refined ranked list.

Hereafter, we denote the set of relevant XML fragments generated in Step (2) as the *Set of Integrated XML Fragments* (*SIXF*). Furthermore, to more readily explain the process of generating the SIXF, we translate each XML fragment into a tree structure. In general, XML documents can be translated into a tree structure by representing tags as nodes in the tree. Each XML fragment in an XML document is associated with a sub-tree in the entire XML tree of the XML document.

2.1 Generating a Set of Integrated XML Fragments

To generate an SIXF, we extract the relevant XML fragments from the simple ranked list generated in Step (1) of our method. Large XML fragments might contain irrelevant fragments and decrease the search accuracy. Therefore, we extract multiple relevant XML fragments from an XML document, as long as their sizes are properly restricted.

One base line approach for generating a nonoverlapped ranked list of XML fragments is to repeatedly extract XML fragments from a simple ranked list in descending order of their rank unless an overlap occurs. The overlapped XML fragments are simply discarded and ignored.

This operation continues as long as either a candidate of the XML fragments remains in the search results or the text size of the extracted XML fragments reaches a predefined upper limit[5].

Next, we aim to reconfigure XML fragments in a simple ranked list to produce results that are better than those of the established base line. We should consider how to identify and extract XML fragments of appropriate lengths in order to attain more accurate XML fragment search. We should also consider how to handle overlapping results, which we ignored in the base line approach.

From the above discussion, we derive the following requirements:

Requirement 1:

Since the traditional search results include several large XML fragments, we should impose an extraction limit on the fragment size.

[5] In our experiments, we extracted 1,000 or less characters for each query. We extract from the beginning of the string value in the text node to the end. When we obtain the first 1,000 characters as a search result, we ignore the rest of the text in the fragment.

Requirement 2:
The extracted XML fragments are appropriately abbreviated and reconstructed to resolve overlap problem.

Extraction Limit To satisfy Requirement 1, we need to limit the size of the extracted XML fragments to an *extraction limit* (EL). Furthermore, we should summarize each XML document, because showing entire documents in search results is not effective. Therefore, we limit the extracted text size for each XML document by defining the EL of an XML document D, whose document ID ($DocID$) is as follows:

$$EL_{DocID} = \alpha \cdot |D_{DocID}| \tag{1}$$

where $|D_{DocID}|$ is the size of the XML document D_{DocID} and α is the ratio of the size of the relevant fragment.

Given this definition, we extract XML fragments from a simple ranked list when the text size of the XML fragments in the SIXF is less than EL. This process repeats until its size exceeds EL.

Reconstructing Fragments To satisfy Requirement 2, we need to arrange the extracted XML fragments so that the SIXF contains useful search results. For generating a non-overlapped ranked list for the base line approach, we simply eliminate the overlapping XML fragments. This may prevent us from extracting relevant XML fragments. For example, in Figure 2, we assume that the XML fragment rooted at node c is the most relevant one in the tree; however, we cannot extract c if we have already extracted d.

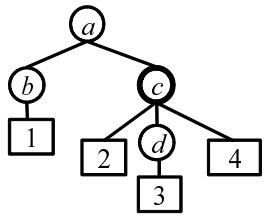

Fig. 2. Example of *overwrite* fragment

To address this problem, we search larger XML fragments and *overwrite* them. As a result, these relevant fragments are all contained in the SIXF, while the existing approaches extract the fragments with the higher score. As these *overwrite* operations are applied, XML fragment lengths in the SIXF increase; therefore, the *overwrite* operation is executed only when Requirement 1 is satisfied.

Again, consider Figure 2. Suppose that c has been extracted. If a is extracted, a overwrites c, because it is larger than c. Furthermore, we discard d, because it is smaller than c.

2.2 Generating a Refined Ranked List

After generating a SIXF, we score each XML fragment in the SIXF and finally generate a refined ranked list. In this process, we aim to obtain informative XML fragments for the higher ranked results. The simplest way to score these XML fragments is to use BM25E, which is used in Step (1) of our method. Unless noted otherwise, we use the BM25E score in the remainder of this paper.

In the following subsections, we propose two scoring methods for generating a refined ranked list: (1) *bottom-up* scoring, which utilizes statistics of descendant XML fragments in order to score an ancestor XML fragment; and (2) *top-down* scoring, which utilizes statistics of an ancestor XML fragment in order to score descendant XML fragments.

Bottom-Up Scoring The *overwrite* operation introduced in Section 2.1 is executed when overlaps exist in the SIXF. In the *overwrite* operation, an ancestor XML fragment is extracted in place of its descendants. The ancestor should be ranked lower in a simple ranked list as compared to its descendants, implying that the refined ranked list also treats the ancestor XML fragment as a lower rank if BM25E is used. As a result, the originally higher-ranked XML fragments cannot always be ranked higher. To score these XML fragments more appropriately, we propose *bottom-up* scoring in order to give more reasonable scores to the XML fragments that contain highly scored descendants.

When we score an XML fragment, we should consider the statistics of its descendant fragments. Conversely, it is not appropriate that XML fragments with low scores are ranked high. Therefore, we must integrate these scores properly. Portions of text nodes in an ancestor XML fragment are composed of descendant XML fragments; the scores of the text nodes in these descendants affect those in the ancestors. Therefore, the *bottom-up* score should be calculated using the BM25E score, as well as the ratio of the lengths of the ancestor XML fragment and that of its descendants.

Let f_a be an ancestor XML fragment and f_d be the descendant fragment with the highest score. We define the *bottom-up* (BU) scoring function as

$$s_{BU}(f_a) = \frac{|f_d|}{|f_a|} \cdot s(f_d) + \frac{|f_a| - |f_d|}{|f_a|} \cdot s(f_a) \tag{2}$$

where $|f_d|$ is the length of f_d, $|f_a|$ is the length of f_a, $s_{BU}(f_a)$ is the *bottom-up* score of f_a, $s(f_a)$ is the BM25E score of f_a, and $s(f_d)$ is the BM25E score of f_d.

Top-Down Scoring Since query keywords may have numerous meanings, it is often difficult to identify a proper one. One solution is to consider the co-occurrence of query keywords. If an XML fragment contains several distinct

query keywords in its text nodes, we can assume the XML fragment to be closely related to the meaning of the given query keywords. In our previous study[5], we obtained informative XML fragments by considering the number of distinct query keywords in each XML fragment.

The larger XML fragments contain more query keywords, indicating that larger XML fragments tend to be ranked higher. In other words, we might overlook smaller XML fragments, although they are informative. To cope with this problem, we propose a scoring method that is independent of XML fragment size.

XML fragments contain numerous distinct query keywords and are identified as informative. Descendant XML fragments of these informative fragments should also be informative. We, therefore, consider *top-down* scoring by calculating the ratio of the number of distinct query keywords contained in an XML fragment to that of its top-level ancestor—i.e., the entire document.

Let f be a scored XML fragment and D_f be an XML document associated with f. We define the *top-down* (TD) scoring function as

$$s_{TD}(f) = s(f) \cdot count(D_f) \tag{3}$$

where $s(f)$ is the BM25E score of f and $count(D_f)$ is the number of distinct query keywords in D_f.

2.3 Example of Generating SIXF and Refined Ranked List

In summary, we illustrate an example of generating a part of an SIXF in which the document ID is 1,000 and its corresponding refined ranked list uses BU scoring. Figure 3 provides a graphical view of this example.

Suppose that $\alpha = \frac{1}{3}$, and $|D_{1,000}| = 300$. Then, $EL_{1,000} = \alpha \cdot |D_{1,000}| = 100$. Next, we introduce $\tau_{1,000}$, which is the total length of the extracted XML fragments from document ID 1,000.

We first obtain a simple ranked list calculated in Step (1). The obtained XML fragments are shown in the left table of Figure 3. For the sake of simplicity, we assume that the list contains only the XML fragments whose document ID is 1,000.

We extract the XML fragment with the highest score, which is node k, from the simple ranked list. Since the text length of k is 40 ($< EL_{1,000}$), k is extracted. This extraction process continues because $\tau_{1,000}$, which contains text node 33, is less than $EL_{1,000}$. Therefore, i is selected next, because i has the second-highest score in the simple ranked list. Thus, i is extracted because $\tau_{1,000}$, which contains text nodes 31 and 33, equals 50 ($< EL_{1,000}$). Node h is the next candidate to be extracted. Since i and k are the descendants of h, they are *overwritten* by h. In particular, i and k are removed from the SIXF and h is added. At this point, $\tau_{1,000}$ becomes 70, which is still less than $EL_{1,000}$.

Next, the *bottom-up* scoring is applied. Equation (2) is used to score node $h(s_{BU}(h)) = \frac{40}{70} \cdot 0.887 + \frac{70-30}{70} \cdot 0.702 = 0.808$.

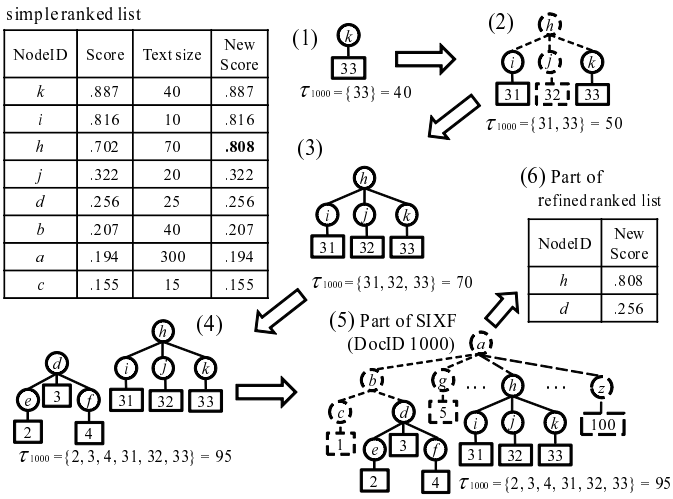

Fig. 3. Processing flow of our method

Following Figure 3 further, d is also extracted. Nodes b and c fail to be extracted because $\tau_{1,000}$ exceeds $EL_{1,000}$. In the end, the SIXF is formed by nodes d and h.

Finally, the refined ranked list is constructed by adding the scores calculated via the *bottom-up* scoring into the SIXF. In the same manner, we generate complete SIXF for all XML documents and construct the final refined ranked list in the descending order of their scores.

2.4 Experimental Evaluation on INEX 2010

We performed our experimental evaluation by using the INEX 2010 test collection. We conducted experiments to compare the previous methods, SIXF,BU, and TD, with the base line approach.

Note that our experiments show that EL, introduced in Section 2.1, does not perform well. As a consequence, we do not apply this approach to the experiments in the remaining subsections.

Table 1 shows that the precisions over the retrieved characters (char_prec) of SIXF are higher than those of the base line (non-overlapped ranked list). TD also overwhelms the base line, and BU decreases its search accuracy when compared with the base line.

As the result indicates, SIXF is the most effective in these methods, though we reported that BU and TD are more effective than SIXF when the INEX 2008 test collection is used. Therefore, we should refine BU and TD scorings if we utilize them for INEX 2010. In next section, we draw a rough sketch of our idea.

Table 1. Comparison of previous methods

		SIXF	*BU*	*TD*	base line
char_prec	(INEX 2010)	.3884	.3277	.3565	.3391
iP[.01]	(INEX 2008)	.6628	.6653	.6637	.6131

3 Investigation of Previous Methods and Exploration of New Methods

Our previous methods[6] tried to extract slightly wider range of relevant fragments. On the other hand, we should try to identify the fragments which can be strongly relevant for a query because our method returns only small sized contents for INEX 2010. Table 2 shows the average number of extracted fragments of each query. It indicates that only one or a few more fragments per a query are extracted. Therefore, we have to carefully determine which fragment is more suitable as a search result because the search accuracy mainly depends on the top-ranked result.

Table 2. Average Number of Extracted Fragments

	SIXF	*BU*	*TD*	base line
average number of extracted fragments	1.212	1.019	1.192	1.192

This suggests that returning the fragments which are the most relevant for a query, i.e., having the highest score, lead us better search results while Section 2.4 shows different. To be concrete, the base line which returns the fragments with the highest score is less effective than SIXF which *overwrites* fragments and extracts a larger fragment.

However, it does not mean that returning larger granular fragments shows higher search accuracy because BU which gives higher score to the larger granular fragments does not overwhelm the base line. In other words, it is not true that search accuracy increases as the granularity of fragments becomes larger[6].

From above discussion, it is more important to reveal the condition of strongly relevant fragments for a query rather than to identify the granularity of them. Therefore, we consider three aspects as follows:

– Two phase extraction
  We find relevant contents after identifying relevant documents. The text size

[6] Actually, the search accuracy of document retrieval is much lower than the base line (char_prec = .2044).

of each fragment affects their score[7]. Therefore, we relax such effect through two steps: 1) identifying a relevant document, and 2) extracting relevant contents to improve search accuracy.

– Management of the range of extraction
  In the step of returning search results in Section 2.4, we extract texts from the beginning of the fragments. However, we can imagine that the range of extracted texts in the fragments influences large effect on search accuracy when we obtain not entire texts in each fragment but part of them. Therefore, we suppose that it is reasonable to consider the condition of each fragment to return appropriate search results.
– Improvement of previous methods
  Some of our previous methods do not perform well as shown in Section 2.4. This is why we try to modify the previous methods along the requirements of INEX 2010. For example, *BU* which utilizes the statistics of descendant fragments emphasizes the ratio of the text sizes of ancestor and descendant fragments. It works well for INEX 2008. However, it does not take effect for INEX 2010 because the granularity of fragments is not so important in 2010. For this reason, we need to refine each scoring method.

4 Related Studies

There are two types of XML documents: (1) data-centric which mainly contain single or compound term in their text nodes and (2) document-centric which tend to contain natural languages in their text nodes[1]. The following subsections describe the existing studies related to both types of XML documents. Although we are primarily interested in search techniques for document-centric XML documents, information on data-centric XML documents will also be useful.

4.1 Data-Centric XML

Data-centric XML documents generally describe only one term in their text nodes. Therefore, studies investigating data-centric XML primarily focus on searching query keywords. The existing research efforts to attain efficient XML fragment search usually utilize the lowest common ancestor (LCA) approach[14]. As a part of this approach, the LCA itself may originate as the top-level common ancestor of arbitrary nodes in an XML tree; however, it is generally defined as the deepest node containing all query keywords in its descendants. Research involving LCA and XML fragments shows significant results related to *efficient* XML search; however, such techniques do not perform well in the context of accurate XML search. In other words, the retrieval accuracy of XML search engines decreases when we use LCAs as the most appropriate XML fragments for a given query[10].

[7] To avoid this effect, BM25E[9] and TF-IPF[8] are normalized by the number of indexed terms.

To address this problem, research efforts have also tried to identify and extract more relevant XML fragments from the sub-tree whose root node is an LCA. XSeek[10] is one such solution, producing a meaningful LCA (MLCA), which classifies and analyzes XML tags by using XML schema information and the positions of query keywords. In the case of XSeek, nodes related to a query are selected and extracted in the order of their relevance to a query. Another approach, eXtract[4], is an expansion of MLCA and infers a user's search purpose by analyzing queries. In eXtract, queries are classified into two cases: (1) extracting an explicit search target and (2) extracting neighbors of the search target.

The purpose of these approaches is similar to that of ours: we wish to return the reconstructed XML fragments. Our method is, however, based on an information retrieval technique for achieving effective XML search, while the abovementioned approaches utilize LCA for the purpose of efficiency.

4.2 Document-Centric XML

Since most document-centric XML documents contain multiple terms in their text nodes, the existing approaches for searching document-centric XML documents focuses on an effective search. In other words, the key objective is to rank more relevant XML fragments higher in the result list. Therefore, conventional information retrieval techniques are often utilized. Another approach is to refine the result list, for example, by removing insignificant fragments or reducing their scores.

Scoring methods for XML fragment search are often derived from the ones used in document search. For example, TF-IPF[2], a popular scoring method for XML fragment search, is an XPath[8]-based scoring method that extends the well-known TF-IDF[13] approach for document search. Another popular approach for XML fragment search is BM25E[9], which is based on Okapi's BM25[12] scoring method for document search.

Although these approaches have been successful, a gap between XML fragment search and document search remains. The goal of XML fragment search is to extract the relevant part of an XML document directly, whereas that of document search is to simply find relevant documents. Extracting relevant XML fragments for a given query is similar to generating snippets in a traditional document search. Snippets, which are short summaries of Web pages that help users judge whether the documents are worth reading, are generated by using information extraction techniques. Since both XML fragments and snippets present useful information to users, such information extraction techniques can be applied to our scoring method.

Manning et al. noted that snippets should have useful information and be maximally informative to a query[11]. To satisfy such requirements, we consider statistics calculated by query conditions[5]; such statistics consist of two components: (1) the ratio of XML fragments containing query keywords amongst XML

[8] `http://www.w3c.org/TR/xpath`

fragments satisfying the constraints of the structure of the given query and (2) the number of query keywords in each XML fragment. In this paper, we denote (1) as the query structure score and (2) as the query keyword score. In our previous study[5], [7], our experiments showed that such methods are more effective than the approaches that do not consider query statistics. We also found that the length of the retrieved XML fragments increases when we use these statistics.

As mentioned above, removing useless or low-scoring fragments is also effective. Although all granularities of XML fragments should be treated as search targets, the effectiveness of the results decreases sharply if a search engine returns noninformative XML fragments. Extremely small XML fragments are often not suitable for search results; Hatano et al. noted that when such meaningless XML fragments are removed, search accuracy improves[3]. Furthermore, our previous study suggests that large XML fragments are also inappropriate for search results[7]. Therefore, we should consider the length of XML fragments.

5 Conclusion

In this paper, we investigate the Ad hoc track of INEX 2010 and drew a rough sketch of our ongoing work. Since our previous methods could not always obtain relevant search results, we should propose a novel approach for returning search results and refine them to adjust to INEX 2010.

References

1. Henk Blanken, Torsten Grabs, Hans-Jörg Schek, Ralf Schenkel, and Gerhard Weikum. *Intelligent Search on XML Data: Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *Lecture Notes on Computer Science*. Springer-Verlag, September 2003.
2. Torsten Grabs and Hans-Jörg Schek. PowerDB-XML: A Platform for Data-Centric and Document-Centric XML Processing. In *Proceedings of the First International XML Database Symposium*, volume 2824 of *Lecture Notes on Computer Science*, pages 100–117. Springer Berlin, September 2003.
3. Kenji Hatano, Hiroko Kinutani, Masahiro Watanabe, Yasuhiro Mori, Masatoshi Yoshikawa, and Shunsuke Uemura. Keyword-based XML Portion Retrieval: Experimental Evaluation based on INEX 2003 Relevance Assessments. In *Proceedings of the Second Workshop of the Initiative for the Evaluation of XML Retrieval*, pages 81–88, March 2004.
4. Yu Huang, Ziyang Liu, and Yi Chen. Query Biased Snippet Generation in XML Search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 315–326. ACM, June/July 2008.
5. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki. A Query-oriented XML Fragment Search Approach on A Relational Database System. *Journal of Digital Information Management (JDIM)*, 8(3):175–180, June 2010.
6. Atsushi Keyaki, Kenji Hatano, and Jun Miyazaki. Result Reconstruction Approach for More Effective XML Fragment Search. In *Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010)*, pages 115–123, November 2010.

7. Atsushi Keyaki, Jun Miyazaki, and Kenji Hatano. A Method of Generating Answer XML Fragment from Ranked Results. In *INEX 2009 Workshop Pre-Proceedings*, pages 563–574, December 2009.
8. Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective Keyword search in Relational Databases. In *Proceedings of the 2006 ACM SIGMOD international Conference on Management of Data*, pages 563–574. ACM, June 2006.
9. Wei Liu, Stephen Robertson, and Andrew Macfarlane. Field-Weighted XML Retrieval Based on BM25. In *Advances in XML Information Retrieval and Evaluation*, volume 3977 of *Lecture Notes on Computer Science*, pages 161–171. Springer Berlin, June 2006.
10. Ziyang Liu and Yi Chen. Identifying Meaningful Return Information for XML Keyword Search. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 329–340. ACM, June 2007.
11. Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval*, pages 157–159. Cambridge University Press, July 2008.
12. Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. *The Third Text REtrieval Conference (TREC-3)*, pages 109–126, 1995.
13. Gerard Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieval. *Journal of Information Processing and Management*, 24(5):513–523, January 1988.
14. Albrecht Schmidt, Martin Kersten, and Menzo Windhouwer. Querying XML Documents Made Easy: Nearest Concept Queries. In *Proceedings of the 17th International Conference on Data Engineering*, page 321. IEEE, April 2001.