Result-Reconstruction Method to Return Useful Results for XML Element Search

Atsushi Keyaki
Graduate School of Culture and Information Science
Doshisha University
1–3 Tatara–Miyakodani
Kyotanabe
Kyoto 610–0394, Japan
keyaki@ilab.doshisha.ac.jp

Kenji Hatano
Faculty of Culture and Information Science
Doshisha University
1–3 Tatara–Miyakodani
Kyotanabe
Kyoto 610–0394, Japan
khatano@mail.doshisha.ac.jp

Jun Miyazaki
Graduate School of Information Science
Nara Institute of Science and Technology
8916–5 Takayama, Ikoma
Nara 630–0192, Japan
miyazaki@is.naist.jp

ABSTRACT

We propose and evaluate a method for obtaining more accurate search results in an extensible markup language (XML) element search, which is a search technique that produces only the relevant elements or portions of an XML document. The existing approaches generate a ranked list in descending order of each XML element's relevance to a search query; however, these approaches often extract irrelevant XML elements and overlook more relevant elements. To address these problems, our approach extracts the relevant XML elements by considering the size of the elements and the relationships between the elements. Next, we score the XML elements to generate a refined ranked list. For scoring, we rank high in the list the XML elements that are the most relevant to the user's information needs. In particular, each XML element is scored using the statistics of its descendant and ancestor XML elements.

Our experimental evaluations show that the proposed method outperforms BM25E, a conventional approach, which neither reconstructs XML elements nor uses descendant and ancestor statistics.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms

PERFORMANCE

Keywords

XML element search, integration and refinement of XML elements, statistics of XML elements

1. INTRODUCTION

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, and we expect this trend to continue in the future. As such, information retrieval techniques for searching XML documents are very important and necessary in the research field of information retrieval.

An XML element is usually defined as part of a larger XML document. The element is identified by its surrounding start and end tags. Therefore, XML elements are nested and have containment relationships with each other, which can cause overlaps in the XML elements in an XML document[1].

Returning search results that contain overlapping XML elements causes users to read the same text multiple times if they check all of the elements. It is likely to be inconvenient for them if they have to read text that has already been presented. Kazai et al. report that the search accuracy[2] of a ranked list that contains elements in descending order of each element's relevance to a search query decreases if XML element search engines do not remove overlapping results [7]. This is the reason why most XML element search engines return a ranked list without overlapping elements. Typically, search engines should return the most appropriate and informative XML elements, because it is said that the search results themselves should be understandable [13]. We define the most appropriate element as one that contains the most relevant text and does not contain irrelevant information in the relevant XML document. In related studies, some XML search engines simply extract the element with the highest score. This means that the other elements are unconditionally discarded as candidates of the search result. Because of this, these kinds of engines do not always return a useful search result, because, even though the search result contains relevant text, too many of its small elements do not make sense in isolation. Moreover, most information retrieval techniques focus on parts of documents where query keywords appear frequently. However, such techniques are not suited for extracting informative elements that satisfy the user's information needs.

[1]Overlapping XML elements are discussed more fully in Section 2.2.
[2]The search accuracy of an XML element search is defined as the ratio of relevant text, while that of a document search is defined as the ratio of relevant documents.

We should judge which XML elements are most appropriate; therefore, we consider the text size of the elements and their containment relationships. We can further identify and improve the precision of XML elements by using statistics derived from the descendant or ancestor XML elements. Combining these approaches, we generate a refined ranked list from a simple ranked list and improve the accuracy and precision of our XML element results.

The rest of this paper is organized as follows. We introduce the basic subject in Section 2 and the related studies in Section 3, followed by a detailed explanation of our approach in Section 4. We also report our experimental results and a discussion of our findings in Section 5. Finally, we conclude our paper in Section 6.

2. PRELIMINARY

In this section, we describe the difference between document search and element search, the definition of elements, and overlapping elements.

2.1 Comparison of Document Search and Element Search

Here, we explain the difference between XML element search engines and well-used Web search engines. When many of the Web search engines return a list of relevant documents, the engines additionally provide users with result snippets [13], which are summaries of each document, approximately 50 words in length. Result snippets are generated by a text extraction technique that extracts the text that is nearby the query keywords. Search-engine users utilize the result snippets located around the result when deciding which documents are worth browsing. Despite the fact that many engines rely on result snippets, not all result snippets help them decide which documents to browse. This is because result snippets do not consider the context; as a consequence, some result snippets do not make sense [14].

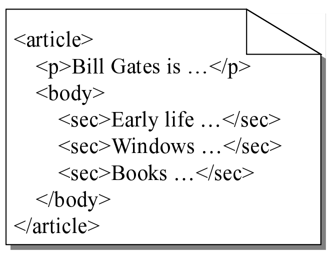
Based on discussions on the Adobe user forums, the users of document search engines should browse the actual documents, because their information needs cannot be satisfied with only result snippets.

On the other hand, the main purpose of an XML element search is to extract the relevant elements from a query and return them in descending order of their relevancy scores. XML element search engines can return a list that contains the relevant parts from a query, while many Web search engines return a list that contains relevant documents for a query. Hence, users do not have to spend time seeking out the relevant parts that satisfy their information needs. This feature saves users' time and energy during information retrieval.

2.2 XML Elements and Their Overlaps

We show concrete examples to explain the definition and overlaps of XML elements.

Figure 1 is an example of an XML document, and Figure 2 is a tree that is translated from Figure 1. XML documents can be expressed as a tree structure, which helps understand the document structure. To provide better understanding, we treat an XML document as a tree structure in this paper. In this regard, a pair of start and end tags represents an XML element node in an XML tree, and a nested structure of XML elements is expressed as their parent-child relationship. Each element in Figure 3 is text that is composed of



```
<article>
  <p>Bill Gates is …</p>
  <body>
    <sec>Early life …</sec>
    <sec>Windows …</sec>
    <sec>Books …</sec>
  </body>
</article>
```

Figure 1: XML document

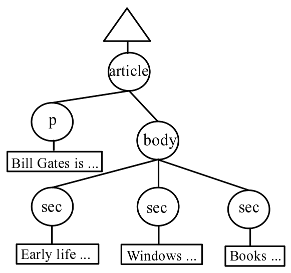


Figure 2: XML tree

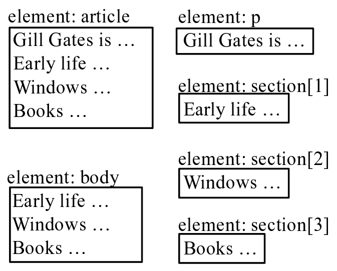


Figure 3: XML element

a set of text nodes of the XML tree in Figure 2. In this concrete example, the article node, which represents the entire document, has all of the text nodes as its descendant text nodes, while the body node also has its descendant text nodes. This demonstrates why there are overlapping XML elements in XML documents. In short, each XML element has a containment relationship, i.e., an ancestor-descendant relationship. Moreover, an ancestor element of an element is called a larger element, and a descendant element of an element is called a smaller element. Finally, we explain informative search results. Suppose that the text nodes "Early life …" and "Windows …", and "Books ⋯" satisfy the user's information needs. In this case, returning the body node to the user is the most appropriate, and this node consists of the informative search results.

2.3 History of XML Information Retrieval

Some existing studies do not remove overlapping elements and return a naively ranked list that is sorted in descending order of the XML elements' scores. We call such a list a **simple ranked list**. On the other hand, most studies have reported damage to search accuracy because of overlapping search results, [7]: therefore, a ranked list without overlapping XML elements is also returned from recent XML element search engines. We call such a ranked list without overlaps a **non-overlapped ranked list**[3]. The INitiative for Evaluation of XML retrieval (INEX) project[4] is the largest ongoing project for XML element search. The INEX project requires search engines to return a non-overlapped ranked list as a search result in XML element searches.

The INEX project carries out an XML element search using document-centric XML documents. The project creates test collections to measure the search accuracy of XML

[3]XML element search engines can extract multiple XML elements if these elements do not overlap with each other.

[4]http://www.inex.otago.ac.nz/

element searches every year. Generally speaking, search engine users browse only the top results in the search results [14]. This means that the most important challenge is to obtain high accuracy within the top-ranked XML elements. For this reason, the INEX project regards iP[.01], which means interpolated precision at recall level 1%, as the formal measure of the evaluation of a search engine. The INEX project also uses mean average interpolated precision (MAiP). MAiP is an evaluation measurement that calculates the average of the (mean) interpolated precision at each recall level. The INEX assessment tool divides the recall levels into 101 levels.

Moreover, the ad hoc track in the INEX project, which tracks effective XML element searches, aims to identify the most appropriate granular element of a search result, because the track tries to reveal the capability of an XML element search. However, document search has been actively studied in recent years, compared with element search, in parts of the INEX project. This is because it is difficult to identify the most appropriate XML element, and it is relatively easier to attain more accurate search results by returning the entire document as an XML element, even though it contains some irrelevant parts. In fact, many of the top-ranked search engines in INEX official results are document search engines [6] over the years. However, one of our goals is to return the most appropriate XML element in one XML document in order to save users' time and effort. Therefore, we attempt to attain accurate searches with an XML element search.

3. RELATED STUDIES

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

3.1 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 element searches usually utilize the lowest common ancestor (LCA) approach [17]. 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 of the query keywords in its descendants. Research involving LCA and XML elements shows significant results related to *efficient* XML search; however, such techniques do not perform well in the context of accurate XML searches. In other words, the retrieval accuracy of XML search engines decreases when we use LCAs as the most appropriate XML elements for a given query [12].

To address this problem, research efforts have also tried to identify and extract more relevant XML elements from the sub-tree whose root node is an LCA. XSeek [12] is one such solution that produces a meaningful LCA (MLCA), which classifies and analyzes XML tags by using XML schema and the positions of the query keywords. In the case of XSeek, the nodes related to a query are selected and extracted in the order of their relevance to a query. Another approach, eXtract [5], 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 the neighbors of the search target.

From the results of the studies, identifying the most appropriate XML element is very difficult, yet important, for XML element search. The purpose of the approaches in these studies is similar to ours to return the best results. However, our method returns not only the LCA, but also any node, because we do not think that the LCA condition is enough for the most appropriate XML element. Moreover, we do not need to utilize foreign information such as XML schema, because we try to identify the appropriate element by using the relationships between the elements.

3.2 Document-centric XML

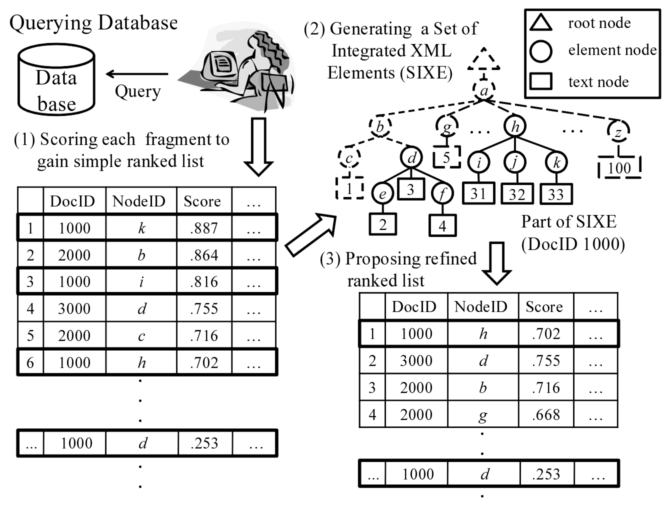
Because most document-centric XML documents contain one or more sentences in their text nodes, the existing approaches for searching document-centric XML documents focus on an *effective* search. In other words, the key objective is to rank the more relevant XML elements higher in the result list. Therefore,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

Scoring methods for XML element search are often derived from the ones used in document search. For example, TF-IPF [3], a popular scoring method for XML element search, is an XPath[5]-based scoring method that extends the well-known TF-IDF [16] approach for document search. Another popular approach for XML element search is BM25E [11], which is based on Okapi's BM25 [15] scoring method for document search.

Although these approaches have been successful, a gap between XML element search and document search remains. The goal of XML element 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 elements for a given query is similar to generating result snippets in a traditional document search. Result snippets are generated by using information extraction techniques. Because both XML elements and result snippets present useful information to users, such information extraction techniques can be applied to our scoring method.

Manning et al. noted that result snippets should have useful information and be maximally informative to a query [13]. To satisfy such requirements, we consider statistics calculated by query conditions [8]. Such statistics consist of two components: (1) the ratio of XML elements containing query keywords among XML elements satisfying the constraints of the structure of the given query, and (2) the number of query keywords in each XML element. In this paper, we denote (1) as the query structure score (*QS*) and (2) as the query keyword score (*QK*). In our previous studies [8, 9], 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 elements increases when we use these statistics.

[5]http://www.w3c.org/TR/xpath


Querying Database
Data base
Query
(1) Scoring each fragment to gain simple ranked list
(2) Generating a Set of Integrated XML Elements (SIXE)
root node
element node
text node
a
b
g
h
z
c
d
i
j
k
e
f
1
2
3
4
5
31
32
33
100
Part of SIXE (DocID 1000)
(3) Proposing refined ranked list

	DocID	NodeID	Score	…
1	1000	*k*	.887	…
2	2000	*b*	.864	…
3	1000	*i*	.816	…
4	3000	*d*	.755	…
5	2000	*c*	.716	…
6	1000	*h*	.702	…
...	1000	*d*	.253	…

	DocID	NodeID	Score	…
1	1000	*h*	.702	…
2	3000	*d*	.755	…
3	2000	*b*	.716	…
4	2000	*g*	.668	…
...	1000	*d*	.253	…

Figure 4: Overview of our proposed method

As mentioned above, removing useless or low-scoring elements is also effective. Although every granularity of XML elements should be treated as search targets, the effectiveness of the results decreases sharply if a search engine returns non-informative XML elements. Extremely small XML elements are often not suitable for search results; Hatano et al. noted that when such meaningless XML elements are removed, the search accuracy improves [4]. Furthermore, our previous study suggests that large XML elements are also inappropriate for search results [9]. Therefore, we should consider the length of the XML elements.

4. SEARCHING FOR XML ELEMENTS

Because users tend to browse only the top search results, we believe it is crucial to attain highly accurate top-ranked results. Similarly, INEX evaluates the precision of the relevant XML elements at a lower recall level.

Search engines should return the smallest-possible XML elements for the given query; however, recently, document search has been regarded as more effective than XML element search (as demonstrated by INEX). We believe that XML element search enables users to save time and energy in their information retrieval tasks and could be very convenient. Therefore, we aim to propose a method to find XML element search results whose length is appropriate.

In our approach, we show a **refined ranked list**, which is composed of the relevant and informative XML elements. As shown in Figure 4, an overview of our method is as follows:

(1) We first score each XML element by using a scoring method to obtain a simple ranked list.

(2) We extract the XML elements from a simple ranked list and generate a set of relevant XML elements by considering the limitations of the XML element length and the reconstruction of the XML elements.

(3) To present a refined ranked list, we rank the XML elements extracted from Step (2); we re-rank and remove the XML elements in the simple ranked list and incorporate them into the refined ranked list.

Hereafter, we denote the set of relevant XML elements generated in Step (2) as the *Set of Integrated XML Elements* (*SIXE*).

4.1 Selecting an Effective Scoring Method

Our method starts with a simple ranked list. Therefore, the search accuracy of the proposed method depends on the quality of the given list. Selecting a suitable scoring method for an XML element search is the key to acquiring a reliable simple ranked list.

We assume the following two conditions:

- A simple ranked list is highly accurate in terms of its MAiP.
- A simple ranked list contains XML elements of varying lengths.

With regard to the first condition, a simple ranked list with high MAiP contains many relevant elements, regardless of whether or not it is ranked in the appropriate order. Therefore, we need to utilize a method with high MAiP.

With regard to the second condition, we want a simple ranked list that is composed of elements of varying lengths, i.e., from small-length elements to large-length elements, because the most appropriate parts in the XML document are not specific granular elements. In other words, the appropriate granularity differs from document to document. Besides, we cannot extract the appropriate granular elements if a simple ranked list contains only specific granular elements. Thus, we assume that a simple ranked list should contain XML elements of varying lengths.

To examine which list satisfies the condition, we utilize the notion that there is a relationship between the granularity and the text size of an element. Note that we consider the ratio of the text size of an element to the granularity of the element. In this case, the standard deviation becomes large if a simple ranked list satisfies the condition. From the above discussion, we will explore the standard deviation for the second condition.

While we try to generate a simple ranked list that contains XML elements of varying lengths, Kamps et al. noted that large-sized XML elements tend to be more effective [6] compared with middle-sized or small-sized ones; however, problems can occur while extracting such large-sized elements. Consider an XML document in which several relevant elements exist. Using the tree structure shown in Figure 4, we consider the sub-trees with the root nodes d and h to be the relevant elements. When we extract a sub-tree that contains all of the relevant elements, it might also include numerous irrelevant elements. For example, if we choose a sub-tree with root node a, it contains numerous irrelevant elements, including c and g. If the scoring method used gives a higher score to large-sized XML elements, result a may rank higher than d and h. To solve this problem, we should extract multiple relevant XML elements from the XML document. In particular, we need to extract differently sized XML elements in order to determine the appropriate granularity of XML elements. Given this approach, we focus on MAiP and the length of the retrieved XML elements. We performed a preliminary experiment, as described in Section 5.2.1, to verify whether or not our assumption is true.

4.2 Generating a Set of Integrated XML Elements

To generate an *SIXE*, we extract the relevant XML elements from the simple ranked list generated in Step (1) of our method. As discussed in Section 4.1, large XML elements might contain irrelevant elements and decrease the search accuracy. Therefore, we extract multiple relevant XML elements from an XML document, as long as their sizes are properly restricted.

One baseline approach for generating a non-overlapped ranked list of XML elements is to repeatedly extract the XML elements from a simple ranked list in descending order of their rank, unless an overlap occurs. The overlapped XML elements are simply discarded and ignored.

This operation continues as long as either a candidate of the XML elements remains in the search results or the number of extracted XML elements reaches a predefined upper limit[6].

On the other hand, we aim to reconfigure XML elements in a simple ranked list to produce results that are better than those of the established baseline. As noted in Section 3.2, we should consider how to identify and extract XML elements of appropriate lengths in order to attain a more accurate XML element search. We should also consider how to handle overlapping results, which we ignored in the baseline approach.

From the above discussion, we derive the following requirements:

Requirement 1:
Because traditional search results include several large XML elements, we should impose an extraction limit on the element size.

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

4.2.1 Extraction Limit

To satisfy Requirement 1, we need to limit the size of the extracted XML elements to an *extraction limit* (EL). Large-sized elements tend to contain more kinds of contents, which means that such elements may have parts that are irrelevant to users' information needs. Thus, we suppose that the text size of the relevant parts in an XML document is lesser than a certain value. We limit the text size of the extracted XML elements for each XML document. To set the limit, we consider two approaches: (1) the value of the limit is dependent on the text size of each document, and (2) the value of the limit is independent of the text size of each document. In approach (1), the document-size-dependent condition, we assume that the relevant parts can be represented as a certain ratio of each XML document size. Accordingly, we limit the extracted text size for each XML document by defining EL of an XML document D as follows:

$$EL_D = \alpha \cdot |D| \qquad (1)$$

where $|D|$ is the size of the XML document D, and α ($0 \leq \alpha \leq 1$) is the ratio of the size of the relevant element.

In approach (2), the document-size-independent condition, there may be XML documents of varying lengths, i.e., there are both large-sized documents and small-sized documents. In this situation, EL derived from approach (1) could be too small a threshold for informative search results if there are extremely small-sized XML documents. Hence, we set EL as a constant value in approach (2) . We define EL as follows:

$$EL_D = \beta \qquad (2)$$

where β is the parameter of the text size, which is the maximum number of relevant parts in an XML document.

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

[6]In our experimentation, we extracted 1,500 or fewer XML elements for each query.

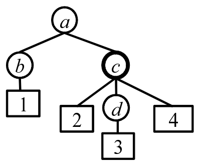
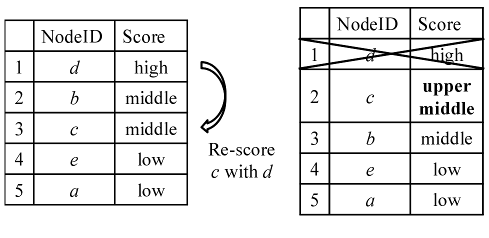


Figure 5: Example of *overwrite* elements



	NodeID	Score
1	*d*	high
2	*b*	middle
3	*c*	middle
4	*e*	low
5	*a*	low

	NodeID	Score
~~1~~	~~*d*~~	~~high~~
2	*c*	**upper middle**
3	*b*	middle
4	*e*	low
5	*a*	low

Figure 6: Overview of Bottom-Up scoring

4.2.2 Reconstructing Elements

To satisfy Requirement 2, we need to arrange the extracted XML elements such that the *SIXE* contains useful search results. To generate a non-overlapped ranked list for the baseline approach, we simply eliminate the overlapping XML elements. This may prevent us from extracting the relevant XML elements. For example, in Figure 5, we assume that the XML element rooted at node c is the most relevant one in the tree; however, we cannot extract c if we have already extracted d.

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

Again, consider Figure 5. 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.

4.3 Generating a Refined Ranked List

After generating an *SIXE*, we score each XML element in the *SIXE* and finally generate a refined ranked list. In this process, we aim to obtain informative XML elements for the higher-ranked results. The simplest way to score these XML elements is to use the initial score, which is used in Step (1) of our method. Unless noted otherwise, we use the initial 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 the statistics of the descendant XML elements in order to score an ancestor XML element, and (2) *top-down* scoring, which utilizes the statistics of an ancestor XML element in order to score the descendant XML elements.

4.3.1 Bottom-Up Scoring

The **overwrite** operation introduced in Section 4.2.2 is executed when overlaps exist in the *SIXE*. In the overwrite operation, an ancestor XML element 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 element as a lower rank if the initial scores are used. In other words, a descendant element that was originally proposed earlier can be proposed later as a part of the ancestor element. We show a concrete example in Figures 5 and 6. When we generate a non-overlapped ranked list with the proposed approach, we first extract d and b. Next, we extract c and remove d from the list because of overwriting. As a consequence, the text nodes in c are proposed as a search result after the text nodes in b are proposed. Regarding the text node in d, it should be proposed before the text node in b is proposed. However, the proposed order of these text nodes is reversed as a result of overwriting. We think that this may damage the search accuracy. Therefore, we define a scoring method in which an element has elements with a higher score among its descendants as a bottom-up scoring method (*BU*).

When we calculate the *BU* score of an XML element, we should consider the statistics of its descendant elements. Conversely, it is not appropriate that XML elements with low scores are ranked high. Therefore, we must integrate these initial scores properly. To re-score a descendant element with the statistics of its ancestor element, we consider two approaches: (1) integrating the scores of the ancestor element and the descendant element by a constant fraction, and (2) integrating the score by a ratio of the lengths of the ancestor XML element and its descendants. Furthermore, we also consider an approach which is a mixture of the former approaches, that is, (3) integrating the scores by a constant ratio after they are re-scored with a ratio of the length of the two elements.

Here, we discuss how to calculate *BU* by (1), a constant ratio. Let f_a be an ancestor XML element and f_d be the descendant element with the highest score. We define the bottom-up (*BU*) scoring function as:

$$s_{BU}(f_a) = \gamma \cdot s(f_d) + (1-\gamma) \cdot s(f_a) \tag{3}$$

where $\gamma (0 \leq \gamma \leq 1)$ is the ratio of the effects on an ancestor element.

Next, we denote the approach for (2). Portions of the text nodes in an ancestor XML element are composed of descendant XML elements; the scores of the text nodes in these descendants affect those in the ancestors. Therefore, the *BU* score should be calculated using the initial score, as well as the ratio of the lengths of the ancestor XML element and that of its descendants:

$$s_{BU}(f_a) = \frac{1}{2}\frac{|f_d|}{|f_a|} \cdot s(f_d) + \frac{1}{2}\frac{|f_a| - |f_d|}{|f_a|} \cdot s(f_a) \tag{4}$$

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 initial score of f_a, and $s(f_d)$ is the initial score of f_d. Note that both initial scores are divided by two to adjust the weight.

Finally, we present an equation to calculate the *BU* score with approach (3), which is a mixture of approaches (1) and (2):

$$s_{BU}(f_a) = \gamma\frac{|f_d|}{|f_a|} \cdot s(f_d) + (1-\gamma)\frac{|f_a| - |f_d|}{|f_a|} \cdot s(f_a) \tag{5}$$

Note that Equation 4 is equal to Equation 5 when $\gamma = 0.5$. We did some experiments to explore the best approach and its parameter of γ in Section 5.4.1.

4.3.2 Top-Down Scoring

Because 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 element contains several distinct query keywords in its text nodes, we can assume that the XML element is closely related to the meaning of the given query keywords. In our previous study [8], we obtained informative XML elements by considering the number of distinct query keywords in each XML element.

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

XML elements contain numerous distinct query keywords and are identified as informative. The descendant XML elements of these informative elements should also be informative. Therefore, we consider top-down scoring method (*TD*) by calculating the ratio of the number of distinct query keywords contained in an XML element to that of its top-level ancestor, i.e., the entire document.

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

$$s_{TD}(f) = s(f) \cdot count(D_f) \tag{6}$$

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

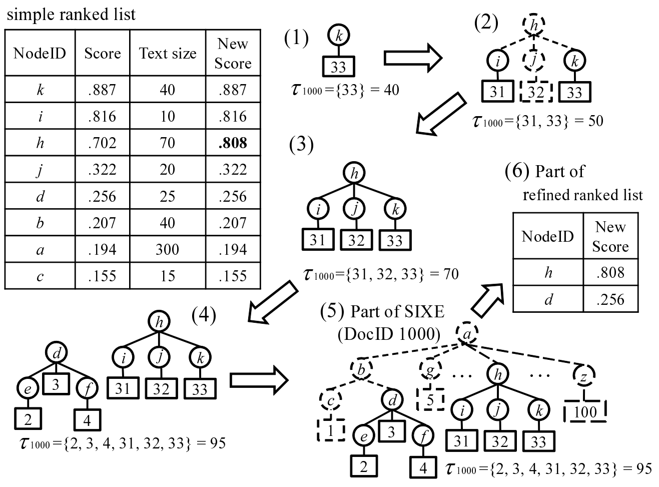
The two methods *BU* and *TD* can be integrated. We call this mixture-scoring method *BU-TD*. We calculate the *BU* score first and then re-score with *TD*, because the *BU* method occurs with overwriting.

4.4 Example of Generating SIXE and Refined Ranked List

In summary, we illustrate an example of generating a part of an *SIXE* in which the document ID is 1,000 and its corresponding refined ranked list uses *BU* scoring. Figure 7 provides a graphical view of this example. Note that we use Equation 4 to calculate the *BU* score.

Suppose that $\alpha = \frac{1}{3}$, and $|D_{1000}| = 300$. Then, $EL_{1000} = \alpha \cdot |D_{1000}| = 100$. Next, we introduce τ_{1000}, which is the total length of the extracted XML elements from document ID 1,000.

We first obtain a simple ranked list that is calculated in Step (1). The obtained XML elements are shown in the left table of Figure 7. For the sake of simplicity, we assume that


simple ranked list

NodeID	Score	Text size	New Score
k	.887	40	.887
i	.816	10	.816
h	.702	70	**.808**
j	.322	20	.322
d	.256	25	.256
b	.207	40	.207
a	.194	300	.194
c	.155	15	.155

(6) Part of refined ranked list

NodeID	New Score
h	.808
d	.256

(1)
τ_{1000}={33} = 40
(2)
τ_{1000}={31, 33} = 50
(3)
τ_{1000}={31, 32, 33} = 70
(4)
τ_{1000}={2, 3, 4, 31, 32, 33} = 95
(5) Part of SIXE (DocID 1000)
τ_{1000}={2, 3, 4, 31, 32, 33} = 95

Figure 7: Example of generating a refined ranked list for an XML document

the list contains only the XML elements whose document ID is 1,000.

We extract the XML element with the highest score, which is node k, from the simple ranked list. Because the text length of k is 40 ($< EL_{1000}$), k is extracted. This extraction process continues, because τ_{1000}, which contains text node 33, is less than EL_{1000}. Therefore, i is selected next, because i has the second-highest score in the simple ranked list. Thus, i is extracted, because τ_{1000}, which contains text nodes 31 and 33, equals 50 ($< EL_{1000}$). Node h is the next candidate to be extracted. Because i and k are the descendants of h, they are overwritten by h. In particular, i and k are removed from the *SIXE* and h is added. At this point, τ_{1000} becomes 70, which is still less than EL_{1000}.

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

Following Figure 7 further, d is also extracted. Nodes b and c fail to be extracted, because τ_{1000} exceeds EL_{1000}. In the end, the *SIXE* is formed by nodes d and h.

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

5. EXPERIMENTAL EVALUATION

5.1 INEX Test Collections

We performed some experiments using two test collections: the INEX 2008 test collection and the INEX 2010 test collection [6, 1].

The INEX test collections consist of three components: (1) the INEX document collection, (2) the INEX topics, and (3) the INEX relevance assessments. The INEX document collection is a Wikipedia XML corpus based on a snapshot of Wikipedia in English. The INEX 2008 test collection collected articles in early 2006, while the INEX 2010 test collection collected articles in late 2008.

The INEX topics include 68 queries. We use all of the topics in the experiments, except for an experiment described in Section 5.5. Each query is represented as narrowed-extended XPath I (NEXI) [18].

The INEX relevance assessments are evaluation tools for XML element search. Using the INEX relevance assessments, an XML search engine can be evaluated on the basis of some evaluation measures by inputting a ranked list into the evaluation tools. We use this in our experimental evaluation. Although the official measure for the focused task in an ad hoc track is iP[.01], we also show MAiP in order to reveal the overall effectiveness of our proposed method.

5.2 Preliminary Experiments

Before evaluating the search accuracy of an *SIXE*, we should choose the scoring method of the initial search in Section 4.1, and also set the parameters α and β for EL in Section 4.2.1. In the next sections, we perform preliminary experiments to find the best scoring method of the initial search and the best parameters with the INEX 2008 test collection. We use the same scoring methods and parameters with the experiments of the INEX 2010 test collection.

5.2.1 *Preliminary Experiments for Scoring Method of Initial Search*

In Section 4.1, we assumed that a simple ranked list that is suitable for *SIXE* has two conditions: (1) a simple ranked list is highly accurate in terms of its MAiP, and (2) a simple ranked list contains many sizes of XML elements. Accordingly, we performed two preliminary experiments to reveal the most suitable scoring method and to evaluate whether or not our assumption is true. In the former experiment, we compared the search accuracy of the following two scoring methods: the normalized TF-IPF [10] and BM25E [11]. Because BM25E requires a weight to be assigned to each tag, we simply set the weight of all tags to 1. The parameters for BM25E are tuned with the INEX 2008 test collection ($k_1 = 2.5, b = 0.85$[7]) and used for both test collections.

Table 8 shows that both MAiP and the standard deviation of BM25E are higher than those of the normalized TF-IPF. Thereby, BM25E should be a more suitable scoring method for our method *SIXE*, if our assumption is valid. To confirm it, we apply *SIXE* to BM25E and the normalized TF-IPF. The result of the experiment is illustrated in Figure 8. The figure shows that the iP[.01] of BM25E (*SIXE*) is higher than that of the normalized TF-IPF (*SIXE*), and the improvement rate between each scoring method and *SIXE* is also higher in BM25E. This suggested that the scoring method returning variously sized XML elements is suitable for our goal, because such a scoring method can return a more appropriate granularity of XML elements when an overwrite operation runs. Therefore, we conclude that the length of the XML elements in a simple ranked list affects the search accuracy of *SIXE*. Note that we use BM25E as a scoring method for the initial search in the following experiments.

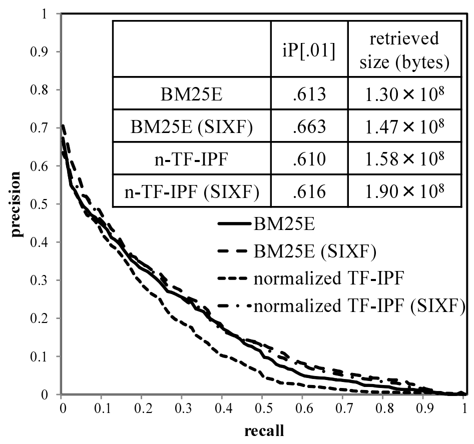
5.2.2 *Preliminary Experiments for EL*

We performed other preliminary experiments to determine the parameter EL, as described in Section 4.2.1, before evaluating our method. First, we performed experiments to evaluate the effectiveness of EL on the document-size-dependent condition, which assumes that the ratio of the relevant parts

[7]The weight of term j in an element i is calculated as follows: $\frac{(k_1+1)tf_{i,j}}{k_1((1-b)+b\frac{el}{avel})+tf_{i,j}} \log \frac{N-df_i+0.5}{df_i+0.5}$ [11].

Table 1: Standard deviation and effect of *SIXE*

	BM25E	normalized TF-IPF
MAiP	.1679	.1399
Standard deviation	1.41×10^{-3}	1.30×10^{-3}



	iP[.01]	retrieved size (bytes)
BM25E	.613	1.30×10^8
BM25E (SIXF)	.663	1.47×10^8
n-TF-IPF	.610	1.58×10^8
n-TF-IPF (SIXF)	.616	1.90×10^8

Figure 8: Comparison of scoring methods

in an XML document is lower than a certain value. Because EL depends on α, we evaluate it with iP[.01] and MAiP by changing α from 0.1 to 1 by steps of 0.1 in the experiment. Table 2 shows iP[.01] and MAiP at α. In this regard, α is the ratio of the relevant parts in an XML document. The experiment shows that the best value of α is 1.0, because iP[.01] and MAiP are the best. This fact indicates that a size limitation on the document-size-dependent condition is not appropriate. Moreover, the result also shows that some XML documents should be returned as a whole document.

We subsequently performed a preliminary experiment to explore the search accuracy of EL on the document-size-independent condition, which assumes that the relevant text in an XML element is lower than a constant value. Table 3 shows iP[.01] and MAiP at β. In this regard, β is the text size of the relevant parts in an XML document. It shows that iP[.01] is the highest when $\beta = 1,000$. From this result, we use EL on the document-size-independent condition with $\beta = 1,000$.

5.3 Evaluation of SIXE

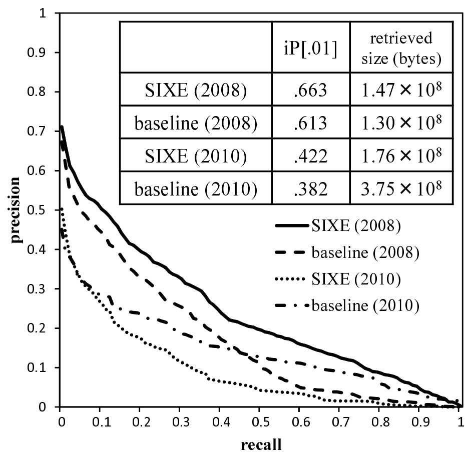
As we discussed in Section 5.2.2, the reconstruction of XML elements is very effective for XML element search. We conducted experiments to compare our method, *SIXE*, with the aforementioned baseline approach. Figure 9 shows that

Table 2: iP[.01] at each α

α	0.1	0.2	0.3	0.4	0.5
iP[.01]	.447	.548	.626	.605	.595
MAiP	.0291	.0505	.0768	.0950	.108
α	0.6	0.7	0.8	0.9	1.0
iP[.01]	.603	.601	.609	.600	.662
MAiP	.123	.132	.138	.140	.237

Table 3: iP[.01] at each β

β	100	200	300	400	500	600	700
iP[.01]	.482	.536	.595	.629	.649	.648	.649
β	800	900	1000	2000	3000	4000	5000
iP[.01]	.655	.656	**.663**	.659	.659	.660	.661



	iP[.01]	retrieved size (bytes)
SIXE (2008)	.663	1.47×10^8
baseline (2008)	.613	1.30×10^8
SIXE (2010)	.422	1.76×10^8
baseline (2010)	.382	3.75×10^8

Figure 9: Comparison of our reconstruction method versus the baseline

all of the interpolated precisions at each recall level of *SIXE* (2008) are higher than those of the baseline (2008). iP[.01] of *SIXE* (2008) is improved by 7% compared with that of the baseline (2008).

Next, we discuss the result related to the INEX 2010 test collection. The interpolated precisions at lower recall levels including iP[.01] of *SIXE* (2010) are higher than that of the baseline (2010). In addition, iP[.01] of *SIXE* (2010) is improved by 10% compared with that of the baseline (2010). As a result, we improved the search accuracy by reconstructing the elements and limiting the extracted text size from each XML document.

However, as the recall level increases, the interpolated precision of *SIXE* decreases sharply using the INEX 2010 test collection. Moreover, the retrieved text size is decreased considerably. These circumstances indicate that EL surely prevents search engines from extracting irrelevant elements, but there are some relevant parts that are actually relevant elements.

5.4 Experiments Related to a Refined Ranked List

In this section, we evaluate the effectiveness of the refined ranked list in order to construct a refined ranked list, and we can use the approaches for scoring *BU* and *TD* that are described in Section 4.3.

5.4.1 *The Best Approach for* BU

We performed some experiments to explore the most effective approach for *BU*. We proposed three approaches for calculating the *BU* score in Section 4.3.1. The approaches are as follows: (1) integrating the scores of the ancestor element and the descendant elements by a constant fraction, (2) integrating the scores by a ratio of the lengths of the

Table 4: iP[.01] at each γ (Equation 3)

γ	0.0	0.1	0.2	0.3	0.4	0.5
iP[.01]	.398	.409	.427	.452	.479	.498
MAiP	.102	.105	.106	.109	.115	.120

γ	0.6	0.7	0.8	0.9	1.0
iP[.01]	.520	.551	.575	.576	.578
MAiP	.126	.130	.134	.135	.137

Table 5: iP[.01] at each γ (Equation 5)

γ	0.0	0.1	0.2	0.3	0.4	0.5
iP[.01]	.461	.485	.512	.550	.605	.665
MAiP	.108	.115	.126	.142	.165	.189

γ	0.6	0.7	0.8	0.9	1.0
iP[.01]	**.669**	.667	.626	.665	.665
MAiP	.190	.190	.190	.189	.189

ancestor XML element and that of its descendant elements, and (3) combining approaches (1) and (2).

We evaluated the search accuracy by changing γ, which is the ratio of the effect of an ancestor element from 0.0 to 1.0 by 0.1 steps, using Equation 3 (Table 4) and Equation 5 (Table 5). We do not have to examine the search accuracy of Equation 4 individually, because the result of Equation 4 is equal to that of Equation 5 when $\gamma = 0.5$.

The results of the two tables show that each iP[.01] of approach (3) is higher than that of approach (2); besides, the iP[.01] of approach (3) with $\gamma = 0.6$ is higher than that of $\gamma = 0.5$. On the basis of these results, the most effective approach for *BU* is calculated by approach (3), which integrates the scores by a constant ratio after re-scoring with a ratio of the length of the two elements with $\gamma = 0.6$. Therefore, we use Equation 5 with $\gamma = 0.6$ in the latter experiments with the two collections.

5.4.2 Effectiveness of Scoring Methods for a Refined Ranked List

We move to the experiments for a refined ranked list. As summarized in Table 6, both of these methods, as well as a mixture of the two methods (*BU-TD*), improved the search accuracy versus *SIXE* in the INEX 2008 test collection. Note that *TD* increased significantly the size of the retrieved XML elements in a ranked list. This is an unpleasant result, because we believe that the XML element search should produce search results that are small rather than large, because larger should const to usets. Conversely, *BU* and *BU-TD* did not increase the size of the retrieved XML elements as much. Furthermore, *BU-TD* can search most accurately among these scoring methods.

Therefore, *BU-TD* is the most suitable method for our proposal. Finally, the iP[.01] of the proposed method is improved by 9% compared to that of the baseline.

On the other hand, only *BU* could improve the search accuracy in the INEX 2010 test collection. There are two possibilities why the search accuracy of *TD* decreased. First, it is not always true that an element in an XML document that contains many kinds of query keywords is effective. Second, we could not generate good *SIXE*, and it may contain some irrelevant elements. From the fact that both the iP[.01] and the MAiP of the baseline in the INEX 2010 test collection are lower than that in the INEX 2008 test collection, the simple ranked list may contain some irrelevant elements. Therefore, we should remove such irrelevant elements from the simple ranked list to tackle the lower accuracy compared with the INEX 2008 test collection.

Furthermore, we compared our approach with other approaches in which INEX plays a part. Table 7 compares our proposed method (*BU-TD* with the INEX 2008 and *BU* with the INEX 2010) with three other competitive XML search engines [6]. Only four engines (including ours) were evaluated in the INEX 2008 test collection. In the experiments, our proposed method provides the highest precision when the recall level is less than or equal to iP[.01]. Because the official measure for the focused task is iP[.01], our engine has the highest level of accuracy.

We also compared our method with the other top-three teams in the INEX 2010 test collection in [1]. The result shows that our method has a higher score in iP[.01] [8].

5.5 Comparisons to Document Search

We finally compared our method (*BU-TD* with the INEX 2008 and *BU* with the INEX 2010) with a traditional document search, because a document search is often reported as being more effective than an XML element search [6]. In a document search, a set of the entire XML documents is returned as a search result generated by BM25E. In this experiment, we used all queries that return the entire XML documents in both of the INEX test collections.

As shown in Figure 10, the results of our experiments showed that all of the interpolated precisions at each recall level of the method *BU-TD* (2008) are higher than those of the document search (2008). This indicates that an XML element search is more effective than a document search. On the other hand, *BU* (2010) also overwhelmed the document search (2010) in recall levels including iP[.01]. Therefore, we conclude that an XML element search can be useful.

Furthermore, we note that the retrieved text size of our proposed method is substantially smaller than that of a document search. As such, we present more focused content to users, which saves them time and energy.

6. CONCLUSION

In this paper, we proposed a method for improving the effectiveness of XML element search. We attempted to identify XML elements that are relevant to a given query. We scored only these XML elements in order to generate a refined ranked list.

The experimental results show that our method is more effective than the traditional approaches, one of which we used as a comparative baseline. We also found that the accuracy of an XML element search can be improved by reconstructing the XML elements and emphasizing the informative ones by applying the statistics of the descendant XML elements. We also keep to a minimum in increasing the text size of the retrieved XML elements, implying that users do not have to sift through larger elements to find the information they are

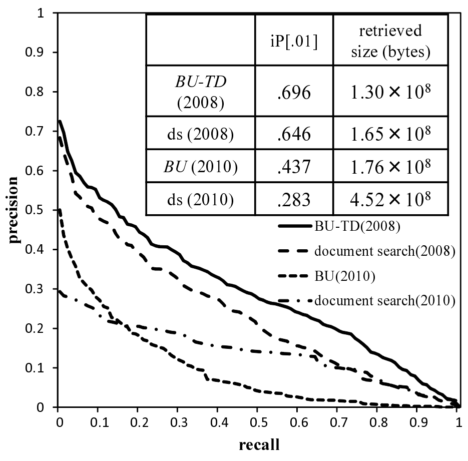
[8]INEX official runs requires search engines to extract 1,000 characters per a query in INEX 2010. To compare with the other systems, we show the search accuracy of the proposed method that extracts 1,000 characters per a query. This is the reason why the result is very different from Table 6.

Table 6: Effect of scoring methods INEX 2008 and INEX 2010

INEX 2008	*BU*	*BU-TD*	*TD*	*SIXE*	baseline
iP[.01]	.668	**.669**	.662	.660	.613
MAiP	.191	.196	.242	.240	.171
retrieved size (bytes)	1.43×10^8	1.43×10^8	2.28×10^8	1.43×10^8	1.30×10^8
INEX 2010	*BU*	*BU-TD*	*TD*	*SIXE*	baseline
iP[.01]	**.437**	.410	.384	.422	.382
MAiP	.0952	.117	.138	.0930	.144
retrieved size (bytes)	1.76×10^8	1.57×10^8	3.68×10^8	1.76×10^8	3.75×10^8

Table 7: Comparison of four INEX participant search engines including our proposed engine

INEX 2008	iP[.00]	iP[.01]	iP[.05]	iP[.10]	MAiP
Doshisha Univ.	.7092	**.6691**	.5606	.5046	.2417
Renmin Univ. of China	.5969	.5969	.5815	.5439	.2486
Queensland Univ. of Technology	.6232	.6220	.5521	.4617	.2134
Univ. of Amsterdam	.6514	.6379	.5901	.5280	.2261
INEX 2010	char prec	iP[.01]	iP[.05]	iP[.10]	MAiP
Doshisha Univ.	.3884	**.1822**	.0382	.0000	.0088
Univ. Pierre et Marie Curie	.4125	.1012	.0385	.0000	.0076
Univ. of Helsinki	.3435	.1186	.0273	.0000	.0069
Lia Univ. of Avignon	.3434	.1500	.0000	.0000	.0077



	iP[.01]	retrieved size (bytes)
BU-TD (2008)	.696	1.30×10^8
ds (2008)	.646	1.65×10^8
BU (2010)	.437	1.76×10^8
ds (2010)	.283	4.52×10^8

Figure 10: Comparison of XML element search and document search

searching for. Furthermore, we found that the search accuracy of our method depends on the scoring function used to generate an initial simple ranked list. In our experiments, BM25E is the most effective scoring method, because it gives high scores to variously sized XML elements.

We set the threshold of EL manually in this article. Therefore, one of our future challenges is to automatically decide the best parameters of EL. Another challenge is that the proposed method gets harmful influence from the irrelevant elements in a simple ranked list. This means that we should consider an approach to remove the irrelevant parts in a simple ranked list to improve search accuracy.

7. REFERENCES

[1] Paavo Arvola, Shlomo Geva, Jaap Kamps, Ralf Schenkel, Andrew Trotman, and Johanna Vainio. Overview of the INEX 2010 Ad Hoc Track. In INEX 2010 Workshop Pre-proceedings, pages 11–40, December 2010.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] Jaap Kamps, Shlomo Geva, Andrew Trotman, Alan Woodley, and Marijn Koolen. Overview of the INEX 2008 Ad Hoc Track. In INEX 2008 Workshop Pre-proceedings, pages 1–28, December 2008.

[7] Gabriella Kazai, Mounia Lalmas, and Arjen P. de Vries. The Overlap Problem in Content-Oriented XML Retrieval Evaluation. In Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 72–79, July 2004.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. Introduction to Information Retrieval, pages 157–159. Cambridge University Press, July 2008.

[14] Satoshi Nakamura. Trustworthiness Analysis of Web Search. Journal of Japanese Society for Artificial Intelligence, 23(6):767–774, November 2008.

[15] 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.

[16] Gerard Salton and Christopher Buckley. Term-Weighting Approaches in Automatic Text Retrieval. Journal of Information Processing and Management, 24(5):513–523, January 1988.

[17] 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.

[18] Andrew Trotman and Börkur Sigurbjörnsson. Narrowed Extended XPath I (NEXI). In Advances in XML Information Retrieval, pages 16–40, May 2005.