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.

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 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.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2010 ACM 978-1-4503-0421-4/10/11 ...$10.00.
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 relevance 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 features 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 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.

The INI itiative for Evaluation of XML retrieval (INEX) project 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

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

4http://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[0.1], 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-IDF [9], a popular scoring method for XML element search, is an XPath-based scoring method that extends the well-known TF-IDF [16] approach for document search. Another popular approach for XML element search is BM25 [11], which is based on Olap’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 a given 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. 

http://www.w3c.org/TR/xpath
4. SEARCHING FOR XML ELEMENTS

We first score each XML element by using a scoring method to obtain a simple ranked list. Then, we extract the XML elements from a simple ranked list and generate a set of relevant XML elements by considering the length of the XML elements. Finally, we apply a threshold to the simple ranked list to obtain a refined ranked list.

In our approach, we show a simple ranked list, which is composed of the relevant XML elements. As shown in Figure 1, 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 length of the XML elements.
3. We apply a threshold to the simple ranked list to obtain a refined ranked list.

As mentioned above, for improving the effectiveness of search engines, the top results should be relevant and high in quality. Therefore, we should consider the length of the XML elements.

When selecting an effective set of XML elements, the search engine can utilize query terms and the metadata of the XML elements. Therefore, it is critical to obtain the top-ranked XML elements. Similar to the XEX, we believe it is critical to obtain the top-ranked XML elements. Similarly, XEX, we believe that XEX, we believe that XEX, we believe that...
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 less 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:

\[
EL_D = \alpha \cdot |D|
\]

where \(|D|\) is the size of the XML document D, and \(\alpha (0 \leq \alpha \leq 1)\) 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)

\(^6\)In our experimentation, we extracted 1,500 or fewer XML elements for each query.

![Figure 5: Example of overwrite elements](image)

![Figure 6: Overview of Bottom-Up scoring](image)

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:

\[
EL_D = \beta
\]

where \(\beta\) 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 \(EL\).

### 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, it 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 included 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 the 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 be an ancestor XML element and fa be the descendant element with the highest score. We define the bottom-up (BU) scoring function as:

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

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:

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

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

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

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 $\gamma$ 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 $D_j$ be an XML document associated with $f$. We define the top-down (TD) scoring function as:

$$s_{TD}(f) = s(f) \cdot \text{count}(D_j)$$

where $s(f)$ is the initial score of $f$, and $\text{count}(D_j)$ is the number of distinct query keywords in $D_j$.

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 $\alpha = 0.5$, and $|D_{1000}| = 300$. Then, $EL_{1000} = \alpha \cdot |D_{1000}| = 100$. Next, we introduce $TD_{1000}$, 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
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 ($< E_L_{1000}$), $k$ is extracted. This extraction process continues, because $\tau_{1000}$, which contains text node 33, is less than $E_L_{1000}$. Therefore, $i$ is selected next, because it has the second-highest score in the simple ranked list. Thus, $i$ is extracted, because $\tau_{1000}$, which contains text nodes 31 and 33, equals 50 ($< E_L_{1000}$). 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, $\tau_{1000}$ becomes 70, which is still less than $E_L_{1000}$.

Next, bottom-up scoring is applied. Function (2) is used to score node $b(s_{b}(h) = \frac{0.887}{0.9} - 0.887 + \frac{20.3}{30} - 0.702 = 0.808)$. Following Figure 7 further, $d$ is also extracted. Nodes $b$ and $c$ fail to be extracted, because $\tau_{1000}$ exceeds $E_L_{1000}$. 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 $ipf\cdot 0.1$, 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 $SIXE$, we should choose the scoring method of the initial search in Section 4.1, and also set the parameters $\alpha$ and $\beta$ 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 IPF·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...
Table 1: Standard deviation and effect of SIXE

<table>
<thead>
<tr>
<th></th>
<th>BM25E</th>
<th>normalized TF-IPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIP</td>
<td>.1679</td>
<td>.1399</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>$1.41 \times 10^{-4}$</td>
<td>$1.30 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Figure 8: Comparison of scoring methods

in an XML document is lower than a certain value. Because EL depends on $\alpha$, we evaluate it with $iP[0.1]$ and MAIP by changing $\alpha$ from 0.1 to 1 by steps of 0.1 in the experiment. Table 2 shows $iP[0.1]$ and MAIP at $\alpha$. In this regard, $\alpha$ is the ratio of the relevant parts in an XML document. The experiment shows that the best value of $\alpha$ is 1.0, because $iP[0.1]$ 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 $iP[0.1]$ and MAIP at $\beta$. In this regard, $\beta$ is the text size of the relevant parts in an XML document. It shows that $iP[0.1]$ 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 all of the interpolated precisions at each recall level of SIXE (2008) are higher than those of the baseline (2008). $iP[0.1]$ 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 $iP[0.1]$ of SIXE (2010) are higher than that of the baseline (2010). In addition, $iP[0.1]$ 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[0.1] at each $\gamma$ (Equation 3)

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>0.0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>iP[0.1]</td>
<td>.398</td>
<td>.400</td>
<td>.397</td>
<td>.392</td>
<td>.392</td>
<td>.388</td>
</tr>
<tr>
<td>MAIn</td>
<td>.102</td>
<td>.105</td>
<td>.106</td>
<td>.109</td>
<td>.115</td>
<td>.120</td>
</tr>
</tbody>
</table>

Table 5: iP[0.1] at each $\gamma$ (Equation 5)

<table>
<thead>
<tr>
<th>$\gamma$</th>
<th>0.0</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>iP[0.1]</td>
<td>.461</td>
<td>.485</td>
<td>.512</td>
<td>.550</td>
<td>.605</td>
<td>.665</td>
</tr>
<tr>
<td>MAIn</td>
<td>.108</td>
<td>.115</td>
<td>.126</td>
<td>.142</td>
<td>.165</td>
<td>.189</td>
</tr>
</tbody>
</table>

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

We evaluated the search accuracy by changing $\gamma$, 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[0.1] of approach (3) is higher than that of approach (2); besides, the iP[0.1] 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 consist of useless. 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[0.1] 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[0.1] and the MAIn 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[0.1]. Because the official measure for the focused task is iP[0.1], 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[0.1] 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[0.1]. 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 propose 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 re-constructing 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 looking for.

8INEX official runs require 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

<table>
<thead>
<tr>
<th>INEX 2008</th>
<th>BU</th>
<th>BU-TD</th>
<th>TD</th>
<th>SIEO</th>
<th>baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP[0</td>
<td>1]</td>
<td>.668</td>
<td>.669</td>
<td>.662</td>
<td>.660</td>
</tr>
<tr>
<td>MAP</td>
<td>.191</td>
<td>.196</td>
<td>242</td>
<td>240</td>
<td>.171</td>
</tr>
<tr>
<td>retrieved size (bytes)</td>
<td>$1.43 \times 10^8$</td>
<td>$1.43 \times 10^8$</td>
<td>$2.28 \times 10^8$</td>
<td>$1.43 \times 10^8$</td>
<td>$1.30 \times 10^8$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INEX 2010</th>
<th>BU</th>
<th>BU-TD</th>
<th>TD</th>
<th>SIEO</th>
<th>baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP[0</td>
<td>1]</td>
<td>.437</td>
<td>.410</td>
<td>.384</td>
<td>.422</td>
</tr>
<tr>
<td>MAP</td>
<td>.055</td>
<td>.117</td>
<td>.138</td>
<td>.003</td>
<td>.144</td>
</tr>
<tr>
<td>retrieved size (bytes)</td>
<td>$1.76 \times 10^8$</td>
<td>$1.57 \times 10^8$</td>
<td>$3.68 \times 10^8$</td>
<td>$1.76 \times 10^8$</td>
<td>$3.75 \times 10^8$</td>
</tr>
</tbody>
</table>

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

| INEX 2008 | IP[0|0] | IP[0|1] | IP[0|3] | IP[1|0] | MAP|
|-----------|-------|-------|-------|-------|-----|
| Doshisha Univ. | .7099 | .6691 | .5066 | .5046 | .2417 |
| Renmin Univ. of China | .5969 | .5690 | .5815 | .5439 | .2486 |
| Queensland Univ. of Technology | .6232 | .6290 | .5521 | .4617 | .2334 |
| Univ. of Amsterdam | .6514 | .6379 | .5001 | .5290 | .2261 |

| INEX 2010 | char prec | IP[0|1] | IP[0|3] | IP[1|0] | MAP|
|-----------|-----------|-------|-------|-------|-----|
| 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 |

Figure 10: Comparison of XML element search and document search

7. REFERENCES


