Dynamic Focused Retrieval of XML Documents and Its Evaluation

Toshiyuki Shimizu
Graduate School of Informatics
Kyoto University
shimizu@soc.i.kyoto-u.ac.jp

Masatoshi Yoshikawa
Graduate School of Informatics
Kyoto University
yoshikawa@i.kyoto-u.ac.jp

ABSTRACT

XML information retrieval (XML-IR) systems search for relevant elements in XML documents for given queries. Though XML-IR systems must handle nesting elements, the output of existing systems remains single ranked list. Ranked lists of all relevant elements may contain redundant contents by nestings, whereas single list of focused elements may lose possible benefit of XML-IR. We introduce the concepts of benefit and effort and propose to retrieve focused elements dynamically. The system flexibly retrieves non-overlapping elements which have larger benefit within the effort acceptable to users. To evaluate XML-IR systems with dynamic focused retrieval, we decided to use an upper bound of the benefit that is obtained by the system because we found no unique algorithm can be the optimal and practical solution for the problem. The performance of the system can be observed compared to the upper bound.

1. INTRODUCTION

To retrieve information about a topic from a large quantity of XML documents, a keyword search can be used to retrieve the document fragments that are relevant to the topic. XML information retrieval (XML-IR) systems generally use elements as basic search units. For example, in the case of academic articles marked up in XML, XML-IR systems use elements corresponding to sections, subsections, and paragraphs to construct search results.

One of the advantages of XML-IR systems is that users need to read only small portions of result documents, and XML-IR systems should retrieve higher relevant portions before they retrieve lower relevant portions. A critical focus of XML-IR is how to handle nesting elements. Some result elements may nest other results; for example, a paragraph element may be highly relevant to a query, while the section element that includes the paragraph is moderately relevant to the query.

INEX 2005 [5] defines three element retrieval strategies for evaluating the effectiveness of XML-IR systems. System using the Thorough strategy simply retrieve relevant elements from all elements and rank them in order of relevance. Elements retrieved using the Thorough strategy may overlap due to nestings. System that use the Focused strategy retrieve only focused elements by selecting the element with the highest score in a path and removing overlapping elements. Though the use of the Focused strategy avoids redundancy, it excludes non-focused elements from the results, which means that some of the possible benefits of XML-IR are lost [1] because it retrieves fixed ranked list of focused elements. Systems using the Fetch and browse strategy first identify relevant documents (fetching phase) and then identify relevant elements within a fetched document (browsing phase).

In INEX 2006 [6], Relevant in context task and Best in context task were also introduced. Relevant in context is a similar strategy to Fetch and browse except that it retrieves only focused elements according to their original document order for each article. Best in context searches for the Best Entry Point (BEP) for starting to read an article and retrieves only one best starting point per article. Retrieving elements grouped per article as in Fetch and browse or Relevant in context is another important point to consider, however it is not focus of this paper.

The retrieval strategies in INEX assume that XML-IR systems return a fixed ranked list of elements. Such retrieval strategies are important for system evaluation, however we considered that the style of single ranked list suffer from handling nesting elements. As we mentioned, the results of Thorough may contain redundant contents by nesting elements, or the Focused never retrieves ancestor elements of elements that have already been retrieved and lacks flexibility.

To handle nesting in XML-IR search results, Clarke [1] proposed controlling overlapping by re-ranking the descendant and ancestor elements of the reported element. However the retrieval style is yet fixed ranked list of elements.

To overcome this problem, we propose dynamic focused retrieval. We introduced the concepts of benefit, which is the amount of gain obtained by reading the element, and effort, which is the cost for browsing search results [10]. We supposed that the user interactively decide the amount of effort that can be spent to browse search results, and the system dynamically retrieve relevant elements within the specified effort. The system retrieves elements that provide larger benefit and to obtain more benefit from retrieved elements, nesting elements are removed from the search results since it is considered that there is no increase in benefit from reading

the repeated content. By using this dynamic retrieval style, the systems can retrieve non-overlapping elements flexibly.

The following situation can be considered as one of the actual usage scenarios of our XML-IR systems. A user first specifies a threshold of effort and the system retrieve focused elements for the specified effort. Then, in response to the retrieval result from the system, user interactively adjust the threshold of effort. If s/he thinks the returned set of subdocuments is too much (or too small), s/he will increase (or decrease) the threshold. For any threshold value of effort, the system always retrieves focused non-overlapping elements depending on the value.

We considered that effort consists of reading effort and switching effort. The reading effort is the effort for reading the contents of the result elements, and the switching effort is the effort for switch result items.

By using reading effort, we can directly handle element size. The elements retrieved by XML-IR systems vary greatly in size. A root element that corresponds to the whole document will be large, while other elements may be much smaller. Therefore, the cost of reading the content of a retrieved element is not known beforehand. In general, users of XML-IR systems browse search results from top ranked element to lower ranked elements. Therefore, fast retrieval of high ranked elements is important and there are some researches on top-k search of XML-IR [12, 3]. However, total output size of top-k elements is uncontrollable by simply giving an integer k. We considered using reading effort instead of an integer k is better alternative to control the total output size.

The systems can avoid long list of short elements by considering switching effort. In this paper, the discussions mainly based on only reading effort excluding switching effort, however we can extend the discussions by considering that we need extra effort, that is switching effort, for browsing each element in result lists.

We formalized the problem of maximizing benefit for a given effort when the system is given benefit and reading effort for each element, and defined the concept of search result continuity, which we consider a critical property for a practical system. Because systems based on our scheme retrieve elements flexibly according to the amount of effort, the contents of the elements retrieved as the optimal solution for the small amount of effort may not be included in the contents of the elements retrieved as the optimal solution for the larger effort. We believe a practical system should avoid this type of situation.

We found that the problem of finding an optimal solution for the formalized problem was an variant of the knapsack problem and was NP-hard. We also found that this optimal solution violated search result continuity, and therefore proposed greedy algorithms for the formalized problem. Furthermore, we improved the algorithm that we proposed in [10], and propose a recursive greedy retrieval algorithm.

We need to devise evaluation measure that is not based on the number of relevant results such as traditional Precision and Recall to evaluate XML-IR systems with dynamic focused retrieval, because such systems are based on benefit and effort and disregard the number of retrieved elements.

We assumed that the reading effort could be easily calculated using the length of the corresponding element, and the switching effort is a constant value in this paper. Thus, the performance of a system based on our scheme depends on the benefit calculated by the system. To evaluate a flexible retrieval system in which the result elements change depending on the specified effort, we assumed we could use the actual benefit of each element. Using actual benefit means that the optimal solution for the formalized problem gives a theoretical upper bound of benefit that the system can provide. We considered that the proportion of the total amount of actual benefit that the system provided compared to the upper bound changing the amount of specified effort could be used as a base for evaluation measures. However, as the problem of finding the optimal solution is NP-hard, we decided to use the upper bound of the benefit provided by the optimal solution as the target for comparison. We confirmed the effectiveness of the upper bound and the quality of the upper bound was sufficient for most queries of INEX 2005.

2. BENEFIT AND EFFORT

Our proposals on retrieval and evaluation for XML-IR are based on the concepts of benefit and effort. We basically assumed the following three points. 1) XML-IR systems retrieve elements, that is, partial text fragments in elements or aggregations of multiple elements can not be answers, and 2) users read full contents of retrieved elements, that is, when a element is retrieved and browsed by a user, the user read all descendant contents together. 3) Users do not read only a partial content of retrieved elements, and users read the content of the retrieved element to the end once s/he start reading. We believe these assumptions are also adopted in element retrieval of INEX [2].

2.1 Benefit

We considered users obtain benefit from relevant content to the query the user input. For a given query, the benefit of an element is the amount of gain obtained by reading the element. In this paper, we follow the assumption of traditional IR and XML-IR methods for the sake of simplicity. The target document set describe unique content, that is they do not repeat the same or similar content more than once. This assumption is adopted by traditional IR and XML-IR methods, which may retrieve redundant but different documents that describe the same or similar content. We can also observe the Precision / Recall measure does not consider such document similarity.

Under this assumption, basically, the benefit of an element can be considered to be the sum of the benefit of the child elements. However, when all the child elements are read together, the contents of the child elements may complement each other; hence the benefit of the parent element may be larger than sum of the benefit of the child elements.

In this paper, we assumed that the benefit of an element is greater than or equal to the sum of the benefit of its child elements.

2.2 Effort

We introduced two types of effort to model the actual user behavior. One is the reading effort which users spend in browsing the content of the search result, and the other is the switching effort which users spend in browsing the result items.

2.2.1 Reading Effort

We considered users spend reading effort when they browse the content of search results. The reading effort of an ele-

ment is the cost spent in reading the content of the element. Note that reading effort does not depend on the query and can be calculated based on the element itself. Basically, the reading effort of an element can be considered as the sum of the reading effort of the child elements. However, when all the elements are read together, users may be continuously reading within the same context. Hence, the reading effort of the parent element may be smaller than the sum of the reading effort of the child elements.

In this paper, we assumed that the reading effort of an element is less than or equal to the sum of the reading effort of its child elements.

2.2.2 *Switching Effort*

We considered users spend switching effort when they finished reading one result item and continue to read the next result item.

Intuitively, users need more effort for a result items that consist of many short paragraph elements than a result items that consist of only one section element which is the parent element of the paragraph elements and describes the same content in fact.

If we consider strictly, the switching effort is the value that depend on the relationship between the switched two result items. However, we handle the switching effort as a fixed constant value in this paper for the sake of simplicity.

2.3 b/e Graph

The XML-IR systems that we propose calculate the benefit of elements in the XML document set for a given query, and retrieve result elements. We assumed that users would specify the threshold amount c for effort, and that the system would retrieve elements that had larger benefit within the specified effort.

We assumed that reading the same content repeatedly would not increase the benefit. If a search result contains nesting elements, the system keeps only the eldest ancestor element of the nesting elements, and removes all the other elements of the nesting elements, because the amount of benefit provided remains the same. Therefore, the search results do not contain nesting elements.

Figure 1 shows an example of benefit and reading effort calculated by a system for a query. In Figure 1, the tree structure of an XML document is represented, and benefit and reading effort are shown in the form of benefit/reading effort adjacent to the element. For the sake of simplicity, we did not assume any concrete formula for calculating benefit and reading effort in Figure 1.

We assumed that we need to read the whole content of the element to obtain benefit from the element. That is, we can not obtain partial benefit for partial reading effort. When a system calculates benefit and reading effort for a query as in Figure 1 and does not take switching effort into consideration, if a user specifies a threshold of effort to 15, the element set that maximizes benefit is $\{e_3, e_2\}$, whereas if 20 is specified, the element set that maximizes benefit is $\{e_3, e_7\}$.

We considered system behaviors can be expressed in the form of a graph, which we call a benefit/effort graph (b/e graph, for short). The behavior of a retrieval algorithm is expressed by plotting the total amount of benefit obtained when the threshold c for effort is changed. Figure 3 shows system behaviors of three algorithms in Figure 2. Algorithms A1 and A2 only consider reading effort as effort, whereas A1' consider both reading effort and switching effort. The value of switching effort in this example is 5. For example, algorithm A1 retrieves $\{\phi\}$ when $0 \leq c < 10$, $\{e_3\}$ when $10 \leq c < 20$, and so on.

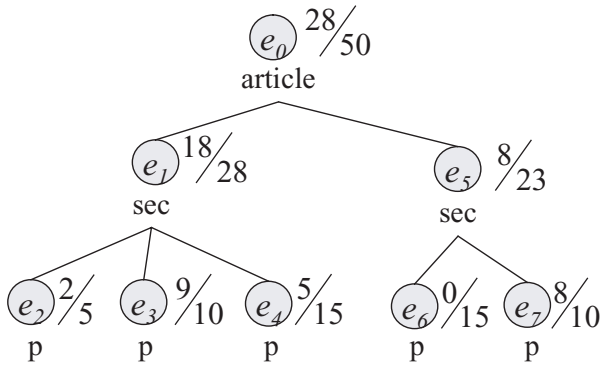

Figure 1: An example of calculated benefit and reading effort.

A1		A2		A1'	
$\{\phi\}$	$[0, 10)$	$\{\phi\}$	$[0, 5)$	$\{\phi\}$	$[0, 10)$
$\{e_3\}$	$[10, 20)$	$\{e_2\}$	$[5, 15)$	$\{e_3\}$	$[10, 25)$
$\{e_3, e_7\}$	$[20, 38)$	$\{e_2, e_3\}$	$[15, 25)$	$\{e_3, e_7\}$	$[25, 43)$
$\{e_1, e_7\}$	$[38, 50)$	$\{e_2, e_3, e_7\}$	$[25, 38)$	$\{e_1, e_7\}$	$[43, 50)$
$\{e_0\}$	$[50, \infty)$	$\{e_1, e_7\}$	$[38, 50)$	$\{e_0\}$	$[50, \infty)$
		$\{e_0\}$	$[50, \infty)$		

Figure 2: Examples of system behaviors.

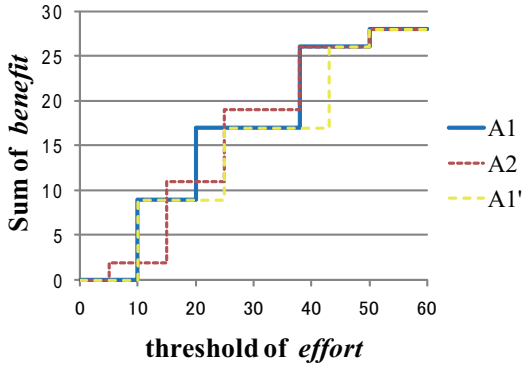

Figure 3: A1, A2 and A1' on b/e graph.

3. RETRIEVAL METHOD

We describe retrieval algorithms based on benefit and effort. First, we handle only reading effort as effort, and then introduce switching effort in Section 3.4. We formalize the problem in Section 3.1, and describe retrieval algorithms in Section 3.2 and 3.3. Finally, we extend the discussion and describe retrieval algorithms considering switching effort in Section 3.4.

3.1 Formalization of the Problem

The problem of maximizing benefit for a given effort is a variant of a knapsack problem [7] that has restriction of nestings. This problem (P) is formalized as follows.

$$
\text{(P)} \quad \left| \begin{array}{ll}
\text{maximize} & z(x) = \sum_{i=1}^{n} b_i x_i \qquad (1) \\
\text{subject to} & \sum_{i=1}^{n} r_i x_i \leq c \qquad (2) \\
& x_i \in \{0,1\} \qquad (3) \\
& x_{j_1} + x_{j_2} + ... + x_{j_m} \leq 1 \qquad (4) \\
& \quad \text{for any } e_{j_1}, e_{j_2}, ..., , e_{j_m} \\
& \quad \text{which are elements} \\
& \quad \text{on a path from root to leaf}
\end{array} \right.
$$

where b_i is the benefit of the element e_i, r_i is the reading effort of e_i, and c is the threshold value of effort input by the user. We can state that e_i is (not) contained in the search result by setting $x_i = 1$ ($x_i = 0$, respectively). The condition (4) shows that the search results do not contain nesting elements.

This problem (P) is considered to be an extension of the normal knapsack problem as it can be reduced to a normal knapsack problem if we handle XML documents that each contains only one element. We can say that the problem (P) is NP-hard since the normal knapsack problem is already NP-hard.

The system that maximizes benefit in the situation shown in Figure 1 retrieves $\{e_3, e_2\}$ when $c = 15$, and $\{e_3, e_7\}$ when $c = 20$. This means that the system does not output the content of e_2 when $c = 20$, though the content of e_2 is output when $c = 15$. Therefore, when a user specifies a threshold of effort to 20, s/he can not obtain information from e_2 though s/he pays more effort than the case when s/he specifies the threshold of effort to 15 and obtains information from e_2. To avoid such situations, we consider it important that systems have the property of search result continuity. Search result continuity is defined as follows.

Definition 1. Search result continuity

When we describe the result element set for threshold c of effort as E^c=$\{e_1^c, e_2^c, ..., e_n^c\}$, and the result element set for the threshold c' as $E^{c'}$=$\{e_1^{c'}, e_2^{c'}, ..., e_m^{c'}\}$, the algorithm has the property of search result continuity if the following holds for any c and c'. The function ancestor-or-self (e) returns an element set that consists of ancestor elements of e and e itself.

If $c \leq c'$, then $\forall\ e \in E^c$, $\exists\ e' \in E^{c'}$ s.t. $e' \in$ ancestor-or-self (e) □

In other words, the content of an element set for effort c must be contained in the content of the element set for effort c' if we increase the threshold value for effort from c to c'. Note that the element e' is unique to all e from Equation 3 and 4 in the context of problem (P). We consider that practical systems must provide search result continuity, as do algorithms A1, A2 and A1' in Section 2.3.

When we consider retrieval algorithms that provide search result continuity, there is no single algorithm that can provide benefit greater than or equal to all the other algorithms for any threshold value of effort in general. In the case of the situation in Figure 1, an algorithm such as A2, which can provide maximum benefit by retrieving $\{e_3$,$e_2\}$ when $c = 15$, can not retrieve $\{e_3, e_7\}$ and provide maximum benefit when $c = 20$.

3.2 Simple Greedy Retrieval Algorithm

As the problem (P) is NP-hard and the optimal solution violates search result continuity, we considered to solve the problem by greedy retrieving elements that we can obtain benefit efficiently.

We considered the Algorithm 1 to provide a greedy solution for (P). The inputs for the algorithm are $list_{in}$ and c. $list_{in}$ is the element list that holds $b_i/r_i \geq b_{i+1}/r_{i+1}$ for all i, and if b_i/r_i is equal to b_{i+1}/r_{i+1}, the list holds $r_i \leq r_{i+1}$. c is the threshold for effort. The outputs of the algorithm are the sum of obtained benefit z and $\boldsymbol{x}$= $\{x_1, x_2, ..., x_n\}$ for z.

When a element is retrieved, by adjusting the benefit and reading effort values of the ancestor elements, and retrieving the element that has the greatest b_i/r_i in that time, the system can retrieve elements efficiently. We call this algorithm a simple greedy retrieval algorithm. The output of this algorithm holds the conditions of (P) and search result continuity.

As an example, we describe a case in which a user specifies a threshold of reading effort c to 40, and benefit and reading effort are calculated as in Figure 1. In this case, $list_{in}$ is $\{e_3(b_3/r_3 = 9/10 = 0.9)$, $e_7(0.8)$, $e_1(0.64)$, $e_0(0.56)$, $e_2(0.4)$, $e_5(0.35)$, $e_4(0.33)\}$. First, e_3 is processed and $\boldsymbol{x}$=$\{e_0, e_1, e_2$, $..., e_7\}$ becomes $\{0,0,0,1,0,0,0,0\}$ and z becomes 9. At the same time, the benefit and reading effort of the ancestor elements e_1 and e_0 are adjusted. For e_1, b_1 is decreased to 9 and r_1 is decreased to 18. For e_0, b_0 is decreased to 19 and r_0 is decreased to 40. The benefit (reading effort) for the ancestor elements after these adjustments is obtained (is required), when the algorithm had a chance to subsequently process the ancestor elements later. The system reflects these adjustments and re-ranks the elements in $list_{in}$. In this case, $list_{in}$ becomes $\{e_7(0.8)$, $e_1(0.5)$, $e_0(0.48)$, $e_2(0.4)$, $e_5(0.35)$, $e_4(0.33)\}$. In addition, c becomes $40 - 10 = 30$. Then, e_7 is processed and $\boldsymbol{x}$ becomes $\{0,0,0,1,0,0,0,1\}$, z becomes $9+8 = 17$, $list_{in}$ becomes $\{e_1(9/18 = 0.5)$, $e_2(0.4)$, $e_0(0.37)$, $e_4(0.33)$, $e_5(0)\}$, and c becomes $30 - 10 = 20$. Next, e_1 is processed and the system sets $x_3 = 0$ because e_3 is the descendant of e_1. In addition, e_2 and e_4 are removed from $list_{in}$. $\boldsymbol{x}$ becomes $\{0,1,0,0,0,0,0,1\}$, z becomes $17 + 9 = 26$, $list_{in}$ becomes $\{e_0(2/12 \simeq 0.17)$, $e_5(0)\}$, and c becomes $20 - 18 = 2$. Next, e_0 is processed, however we can not retrieve e_0 within the specified effort, and the processing terminates . The outputs are $z = 26$ and $\boldsymbol{x}$=$\{0,1,0,0,0,0,0,1\}$.

The line labeled "simple" in Figure 4 shows the system behavior of a system using the simple greedy retrieval algorithm when the system calculates benefit for a query as in Figure 1.

3.3 Recursive Greedy Retrieval Algorithm

When the sum of the reading effort of elements retrieved using the simple greedy retrieval algorithm is less than the threshold value c, it is considered that the amount of benefit is increased by retrieving elements that have not yet been obtained using the remainder of effort.

However, it is important that systems have the property

Algorithm 1 Simple greedy retrieval algorithm

Input: $list_{in}$, c
Output: z, $\boldsymbol{x}$
1: $z = 0$
2: while $((e_i = top(list_{in}))! = null)$ do
3: remove e_i from $list_{in}$
4: if $(r_i > c)$ then
5: break
6: end if
7: $adjust(e_i)$
8: $x_i = 1$, $z+ = b_i$, $c- = r_i$
9: end while
10: return z, $\boldsymbol{x}$
11:
12: function $\boldsymbol{adjust(e_i)}$ {
13: for $(\boldsymbol{e_d \in e_i.descendants})$ do
14: $\boldsymbol{x_d = 0}$
15: $\boldsymbol{remove}$ $\boldsymbol{e_d}$ from $\boldsymbol{list_{in}}$
16: end for
17: for $(\boldsymbol{e_a \in e_i.ancestors})$ do
18: $\boldsymbol{b_a - = b_i}$, $\boldsymbol{r_a - = r_i}$
19: $\boldsymbol{rerank}$ $\boldsymbol{e_a}$ in $\boldsymbol{list_{in}}$
20: end for
21: }

of search result continuity. Therefore, it is not appropriate to obtain any elements for the remainder of effort. To satisfy search result continuity, the system needs to retrieve descendant elements of the element that is to be retrieved next. An algorithm that improves on the simple greedy retrieval algorithm by retrieving more elements for the remainder of effort is shown in Algorithm 2. We call this algorithm a recursive greedy retrieval algorithm.

In the case of the running example, when a user sets $c = 30$, the outputs of the simple greedy retrieval algorithm are $z = 17$ and $\boldsymbol{x}$=$\{0, 0, 0, 1, 0, 0, 0, 1\}$. However, a system using the recursive greedy retrieval algorithm can retrieve e_2 which is a descendant of e_1 after e_7; the outputs of the recursive greedy retrieval algorithm are $z = 19$ and $\boldsymbol{x}$=$\{0, 0, 1, 1, 0, 0, 0, 1\}$.

The line labeled "recursive" in Figure 4 shows the system behavior of a system using the recursive greedy retrieval algorithm when the system calculates benefit for a query as in Figure 1.

Theorem 1. Superiority of recursive greedy retrieval algorithm

The sum of benefit from elements retrieved by the recursive greedy retrieval algorithm is greater than or equal to that provided by the simple greedy retrieval algorithm for any threshold value of effort.

Proof omitted as trivial. We should therefore use the recursive greedy retrieval algorithm in the implementation of systems.

3.4 Consideration for Switching Effort

If we consider switching effort, we need to change the condition 2 in (P) to the following.

$$\sum_{i=1}^{n}(r_i + s)x_i - s \leq c \tag{5}$$

Algorithm 2 Recursive greedy retrieval algorithm

Input: $list_{in}$, c
Output: z, $\boldsymbol{x}$
1: $z = 0$
2: while $((e_i = top(list_{in}))! = null)$ do
3: if $(!retrieve(e_i))$ then
4: break
5: end if
6: end while
7: return z, $\boldsymbol{x}$
8:
9: function $\boldsymbol{retrieve(e_i)}$ {
10: remove $\boldsymbol{e_i}$ from $\boldsymbol{list_{in}}$
11: if $(\boldsymbol{r_i > c})$ then
12: $\boldsymbol{retrieveDescendants(e_i)}$
13: return false
14: end if
15: $\boldsymbol{adjust(e_i)}$
16: $\boldsymbol{x_i = 1}$, $\boldsymbol{z+ = b_i}$, $\boldsymbol{c- = r_i}$
17: return true
18: }
19:
20: function $\boldsymbol{retrieveDescendants(e_i)}$ {
21: $\boldsymbol{e_m}$ = top ranked element that is descendant of $\boldsymbol{e_i}$ and $\boldsymbol{x_m = 0}$
22: if $(\boldsymbol{e_m == null})$ then
23: return
24: end if
25: if $(\boldsymbol{retrieve(e_m)})$ then
26: $\boldsymbol{retrieveDescendants(e_i)}$
27: end if
28: }

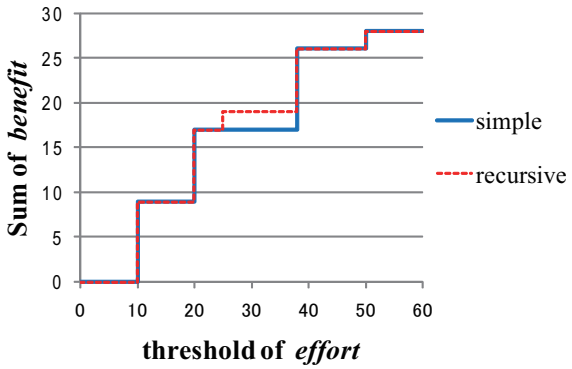

Figure 4: System behaviors of each algorithm on b/e graph.

where s is the switching effort.

We can extend retrieval algorithms in Section 3.2 and Section 3.3 by considering that we must spend additional switching effort to retrieve each element and c is also incremented by s.

We show the recursive greedy retrieval algorithm considering switching effort in Algorithm 3. The effort of each element and c is incremented by s. If we drop the $retrieveDescendants(e_i)$ function, the algorithm can be consider as simple greedy retrieval algorithm.

Algorithm 3 Retrieval algorithm considering switching effort

Input: $list_{in}$, c, s
Output: z, $\boldsymbol{x}$
1: $z = 0$
2: $c+ = s$
3: for ($e_i \in list_{in}$) do
4: $r_i+ = s$
5: end for
6: $e_{top} = top(list_{in})$
7: if ($!retrieve(e_{top})$) then
8: return z, $\boldsymbol{x}$
9: end if
10: $rerank\ list_{in}$
11: while $((e_i = top(list_{in}))! = null)$ do
12: if ($!retrieve(e_i)$) then
13: break
14: end if
15: end while
16: return z, $\boldsymbol{x}$

4. EVALUATION METRICS

We propose evaluation metrics based on the system behavior shown in the b/e graph. In order to evaluate systems based on benefit and effort, we supposed that the actual benefit for each element is available. Though systems calculate benefit for each element and retrieve results using recursive greedy retrieval algorithm, we must use the actual benefit by retrieved results for system evaluation not the benefit value expected by the system. Note that as Figure 3 or Figure 4 is drawn using the benefit values calculated by a system, the actual benefit obtained by the systems may differ from the values in these figures.

We decided to use an upper bound of the actual benefit that is obtained by systems. The performance of the system can be observed compared to the upper bound. We do not consider switching effort distinctly in this section, as we can consider that we need extra switching effort for browsing each element when the we want to take the switching effort into consideration. In this section, first we discuss how to calculate actual benefit and effort in Section 4.1, then describe why and how to calculate and use the upper bound in Section 4.2 and Section 4.3. Finally we compare our metrics with INEX metrics in Section 4.4.

4.1 How to Calculate Actual Benefit and Effort

INEX are manually developing relevance assessments for XML-IR. The relevance assessments of INEX 2005 consists of two parts, Exhaustivity (ex) and Specificity (sp) [1]. Exhaustivity is the extent to which the element discusses the topic of request, and it has three levels; Highly exhaustive (HE), Partially exhaustive (PE), and Not exhaustive (NE) [2]. We can convert HE, PE, and NE to numeric as 1, 0.5, 0, respectively. Specificity is the extent to which the element focuses on the topic of request, and it is calculated by dividing $rsize$, which is the length of the content relevant to the topic, by $size$, which is the whole length of the element.

We describe ex, sp, $rsize$, and $size$ of element e_i as $ex(e_i)$, $sp(e_i)$, $rsize(e_i)$, $size(e_i)$. We considered calculating actual benefit and reading effort from assessments of INEX 2005. For example, we can use following equations.

$$b_i = ex(e_i) * rsize(e_i) \quad (6)$$

$$r_i = size(e_i) \quad (7)$$

We can use only $rsize$ for calculating benefit when the assessments of INEX 2006 or later which include only Specificity are used.

$$b_i = rsize(e_i) \quad (8)$$

We assumed the switching effort is the fixed value in this paper. We need to use reasonable value to integrate with reading effort. Though the switching effort should be carefully investigated through user studies, we supposed that the reading effort corresponding to small but meaningful element size such as average size of paragraph elements is likely to be used as a starting value of switching effort.

4.2 Upper Bound of Benefit for Given Effort

When we consider the problem (P) in Section 3.1 and use actual benefit, we can calculate the maximum of actual benefit value for a certain c value by solving (P). If an algorithm could provide maximum benefit for all of the c values, we could use the benefit values of the algorithm as a basis for system evaluation. However, for this problem, there is no such algorithm because we must also consider search result continuity. We therefore decided to calculate the upper bound of the amount of benefit and use it as the basis for evaluating systems. The optimal solution of (P), which does not consider the search result continuity, can be an upper bound. However, this problem (P) is NP-hard and difficult to solve. Therefore, we decided to calculate an upper bound of (P). The upper bound of (P) is greater than or equal to that of the problem considering search result continuity.

Theorem 2. Upper bound of (P)
The optimal value of the continuous problem (P') of (P) that relaxes the condition (3) to $0 \leq x_i \leq 1$ provides the upper bound of (P).

Proof. The value that can be obtained in (P) can also be obtained in (P') because the problem (P') relaxes the condition in (P). The range of values in (P) is included in the range of values in (P'). Therefore, the optimal value of (P') provides an upper bound of (P). □

We considered the Algorithm 4 based on the Algorithm 1 to provide the optimal solution for (P'). If the system can not retrieve whole e_i (Line 4), it achieves optimal value by retrieving the partial amount that can be retrieved. As an example of calculating optimal value of (P'), we consider the case when a user specifies the threshold of effort c to 40, and benefit and reading effort are calculated like Figure 1, which is the same situation when we consider simple greedy retrieval algorithm. The processing basically follows the same steps except that the last step concerning e_0 is skipped. As the final step, system retrieves partial amount of e_0 using the remaining effort 2. x_0 is set to $2/12 \simeq 0.17$ and x_1 and x_7 are set to $1 - 2/12 \simeq 0.83$. $\boldsymbol{x}$ becomes $\{0.17, 0.83, 0, 0, 0, 0, 0, 0.83\}$, z' becomes $26 + 2 * 0.17 \simeq 26.3$, and then breaks.

When we pick up a non-overlapping element set $\{e_{k_1}, e_{k_2}, ..., e_{k_m}\}$, each of which is the descendant of the element e_r,

[1] The assessments of INEX 2006 or later only use Specificity.
[2] Too Small (TS) is introduced for small elements, however we regard TS is equal to NE.

Algorithm 4 Optimal solution for (P')

Input: $list_{in}$, c
Output: z', $\boldsymbol{x}$

```
1: z' = 0
2: while ((e_i = top(list_in))! = null) do
3:    remove e_i from list_in
4:    if (r_i > c) then
5:       x_i = c/r_i
6:       for (e_d ∈ e_i.descendants) do
7:          if (x_d == 1) then
8:             x_d = 1 − x_i
9:          end if
10:      end for
11:      z' += b_i * x_i
12:      break
13:   end if
14:   adjust(e_i)
15:   x_i = 1, z' += b_i, c− = r_i
16: end while
17: return z', x
```

the conditions in (P') hold if we set $x_r = \alpha$ and $x_{k_i} = 1 - \alpha$ $(1 \leq i \leq m)$. In this situation, the sum of benefit is calculated as follows.

$$\begin{aligned} & b_r * \alpha + \sum_{i=1}^{m} (b_{k_i} * (1-\alpha)) \\ = \; & b_r * \alpha + \sum_{i=1}^{m} b_{k_i} - \sum_{i=1}^{m} (b_{k_i} * \alpha) \\ = \; & (b_r - \sum_{i=1}^{m} b_{k_i}) * \alpha + \sum_{i=1}^{m} b_{k_i} \end{aligned} \quad (9)$$

Similarly, the sum of reading effort is calculated by replacing b_i by r_i in Equation 9. That is, setting $x_r = \alpha$ and $x_{k_i} = 1 - \alpha (1 \leq i \leq m)$ is the same as setting $x_v = \alpha$ and $x_{k_i} = 1$ $(1 \leq i \leq m)$ if we assume a virtual element e_v with benefit of $b_r - \sum_{i=1}^{m} b_{k_i}$ and reading effort of $r_r - \sum_{i=1}^{m} r_{k_i}$.

If the system can retrieve whole e_i, that is, it can set $x_i = 1$, setting $x_d = \alpha > 0$ for descendant element e_d of e_i is contrary to the optimal solution because the benefit obtained becomes $b_i*(1-\alpha)+b_d*\alpha$ and $b_i*(1-\alpha)+b_d*\alpha \leq b_i$ holds as b_i is greater than or equal to b_d. When the system retrieves descendant elements of e_i before retrieving e_i, the situation is considered to be $x_v = 1$ and $x_{k_i} = 1$ when we assume the virtual element e_v, and therefore $x_r = 1$ and $x_{k_i} = 0$, in fact.

4.3 Comparison with Upper Bound

The upper bound is represented in the b/e graph as a linear interpolation line of the plots when elements are retrieved by a simple greedy retrieval algorithm. A system using the simple greedy retrieval algorithm can obtain maximum benefit at such points.

For system evaluation, we need the actual benefit for each element. Note that this actual benefit can not be used in system implementation. Implementers of XML-IR systems can develop better systems by guessing the benefit of each element as closely as possible to the actual benefit. We assumed that we could use a common reading effort value for various systems because this value does not depend on the query.

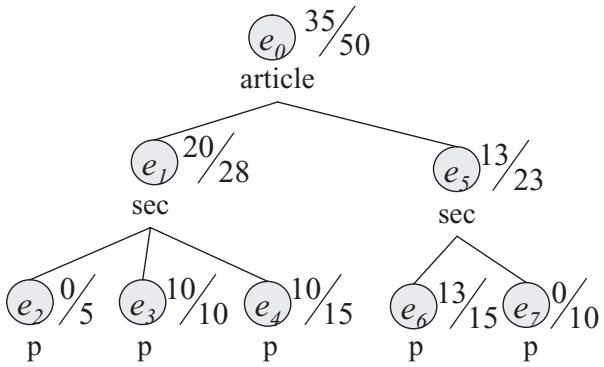

Figure 5: Actual benefit and reading effort.

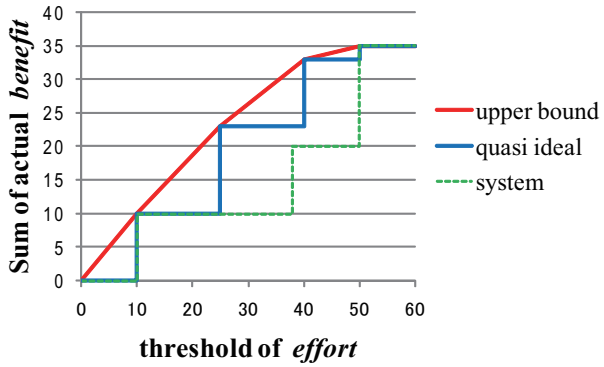

Figure 6: b/e graph for evaluation.

To evaluate systems based on benefit and effort, we can compare the behavior of an implemented system with the behavior of the upper bound on a b/e graph. If we implement a system that can calculate benefit values for each element that are the same value as the actual benefit, a system using the recursive greedy retrieval algorithm can provide benefit that is very close to the upper bound. We call such a system a quasi ideal system.

As an example, we explain the case in which the system calculates benefit and reading effort as in Figure 1, however the actual benefit and reading effort are those shown in Figure 5. In this case, using the recursive greedy ranking algorithm, with a threshold value of effort 40, the system can obtain 20 benefit by $\boldsymbol{x}$=$\{0, 0, 1, 1, 0, 0, 0, 1\}$, however the upper bound of benefit that can be obtained is 33 by $\boldsymbol{x}$=$\{0, 0, 0, 1, 1, 0, 1, 0\}$. Figure 6 shows a b/e graph for the running example. In Figure 6, 'upper bound' is for the upper bound and 'system' is for the system to be evaluated. If the system calculates benefit identical to the actual benefit, the behavior of the system is the line labeled 'quasi ideal'. The b/e graph enables us to intuitively understand the performance of the system in relation to the upper bound. By using the upper bound for comparison target, we can evaluate systems absolutely.

To evaluate XML-IR systems, the metrics based on XCG is used in INEX 2005 [4]. By considering the effectiveness

of the implemented systems in relation to the upper bound, we can apply this concept to our case. For example, we can introduce iMArep (interpolated Mean Average reading effort precision), which is calculated based on a b/e graph using the similar concept of iMAep (interpolated Mean Average effort precision) [4].

We calculated iArep values for quasi ideal system using INEX 2005 test collection. We used calculation formula for actual benefit and reading effort in Section 4.1 and disregarded switching effort. We obtained b/e graphs up to 50,000 effort, which means 50,000 characters that is corresponding to about the length of one article of INEX. Most of the iArep values for quasi ideal system were greater than 0.9, indicating that the quality of the upper bound was good. The iMArep value was 0.90 for all 29 Topics and 0.93 when we excluded Topics 209, 217, and 239 whose iArep value was relatively low because the good results tended to be large elements for such topics.

4.4 Metrics of INEX

Though the target of our metrics and that of INEX is differ, our evaluation metrics is motivated by that of INEX. Our metrics is for dynamic focused retrieval, whereas metrics of INEX is for single ranked list.

Though the evaluation metrics for XML-IR systems is still a task with several open issues, the metrics based on eXtended Cumulated Gain (XCG) is used in INEX 2005 [4]. System-oriented ep/gr (effort-precision/gain-recall) and user-oriented nxCG (normalized extended Cumulated Gain) measures are used by considering the relative effectiveness for the ideal system.

In the context of XCG based measures, effort is measured in terms of the number of visited ranks, whereas our reading effort is measured in terms of element length. In addition, such ideal system in XCG does not exist for the dynamic focused retrieval, and we decided to use the upper bound.

In INEX 2007, evaluation metrics called HiXEval based on relevant text length is used [8, 9]. The HiXEval metrics is based on the assumption that a system which retrieves elements that contain as much relevant text as possible, and as little irrelevant text as possible is preferred. In the HiXEval metrics, the Precision is measured by the length of retrieved relevant text compared to the total length of retrieved text, and the Recall is measured by the length of retrieved relevant text compared to the total length of relevant text.

If we regard element size as reading effort and relevant text size as benefit, the concept of HiXEval is similar to our metrics. In fact, element size can be seen as the factor of reading effort and relevant text size as that of benefit as in Section 4.1. However HiXEval does not consider ideal system as in XCG. This means that though HiXEval metrics can evaluate systems relatively, it can not evaluate systems absolutely. If the value obtained using HiXEval metrics is low, we can not say the system is ineffective, because the value may be low even for an ideal system. To evaluate systems absolutely, we need an ideal system for comparison. In our scheme, we used the upper bound of the optimal solution as the ideal value for comparison

5. CONCLUSIONS

We introduced the concepts of benefit and effort for XML-IR systems, and proposed a retrieval algorithm and evaluation metrics based on them. We examined situations in which users of XML-IR systems specify a threshold for effort and the system flexibly retrieves focused elements dynamically within the specified effort. We formalized the problem and calculated the upper bound of benefit for system evaluation.

In general, existing XML-IR systems calculate relevance score between each element and the input query using some scoring formula, and the length (in characters or in number of terms) of the element is included as a factor of the scoring formula [12], and scores are decreased for longer elements to avoid overestimation of longer elements. By handling element length separately as reading effort, we believe we can apply our scheme to other systems.

In future work, we will examine ways of distinguishing the portion of the retrieved element to be read. Furthermore, for XML documents created by marking up original PDF files, there is potential to show search result elements mapped on an image of a physical page [11]. We will look at ways of integrating our system with this type of user interface. A major drawback of our current scheme is that users must specify the threshold of effort. We believe that developing user interfaces that can smoothly retrieve result elements when users change the threshold value of effort is a promising solution. When we consider about the user interfaces, we can also imagine the opposite case in which users change the threshold value of benefit and the system retrieves elements which minimize effort.

6. REFERENCES

[1] C. L. A. Clarke. Controlling overlap in content-oriented XML retrieval. In SIGIR, pages 314–321, 2005.

[2] INEX. INitiative for the Evaluation of XML Retrieval. http://inex.is.informatik.uni-duisburg.de/.

[3] R. Kaushik, R. Krishnamurthy, J. F. Naughton, and R. Ramakrishnan. On the integration of structure indexes and inverted lists. In SIGMOD, pages 779–790, 2004.

[4] G. Kazai and M. Lalmas. INEX 2005 evaluation measures. In INEX, pages 16–29, 2005.

[5] S. Malik, G. Kazai, M. Lalmas, and N. Fuhr. Overview of INEX 2005. In INEX, pages 1–15, 2005.

[6] S. Malik, A. Trotman, M. Lalmas, and N. Fuhr. Overview of INEX 2006. In INEX, pages 1–11, 2006.

[7] S. Martello and P. Toth. Knapsack problems: algorithms and computer implementations. John Wiley & Sons Inc, New York, 1990.

[8] J. Pehcevski, J. Kamps, G. Kazai, M. Lalmas, P. Ogilvie, B. Piwowarski, and S. Robertson. INEX 2007 evaluation measures. In INEX 2007 Pre-Proceedings, 2007.

[9] J. Pehcevski and J. A. Thom. HiXEval: Highlighting XML retrieval evaluation. In INEX, pages 43–57, 2005.

[10] T. Shimizu and M. Yoshikawa. A ranking scheme for XML information retrieval based on benefit and reading effort. In ICADL, pages 230–240, 2007.

[11] T. Shimizu and M. Yoshikawa. XML information retrieval considering physical page layout of logical elements. In WebDB, 2007.

[12] M. Theobald, R. Schenkel, and G. Weikum. An efficient and versatile query engine for TopX search. In VLDB, pages 625–636, 2005.