A Ranking Scheme for XML Information Retrieval Based on Benefit and Reading Effort

Toshiyuki Shimizu and Masatoshi Yoshikawa

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

Abstract. XML information retrieval (XML-IR) systems search for relevant document fragments in XML documents for given queries. In top-k search, users control the size of output by an integer k. In XML-IR, however, each output element varies widely in size. Consequently, total output size of top-k elements is uncontrollable by simply giving an integer k. In addition, search results may have nesting elements. If a system orders result elements simply by their relevance, we may browse the same content more than once due to the nestings. To handle these problems, we propose a new ranking method that enables us to browse search results of XML-IR systems efficiently by introducing the concepts of *benefit* and *reading effort*. We also propose an evaluation metrics based on *benefit* and *reading effort*, and compared the metrics with existing XML-IR metrics by experiments.

1 Introduction

When we want to retrieve information about a topic from large amount of XML documents, keyword search is one solution to retrieve document fragments relevant to the topic. XML information retrieval (XML-IR) systems generally use elements as search units, and output ranked elements relevant to given queries. For example, in the case of scholarly articles marked up in XML, XML-IR systems retrieve and rank such elements corresponding to sections, subsections, and paragraphs.

INEX 2005 [1] defines three element retrieval strategies for the purpose of evaluating the effectiveness of XML-IR systems. A system with the *Thorough* strategy simply retrieves relevant elements from all elements and ranks them in order of relevance. The retrieved elements using the *Thorough* strategy may overlap due to nestings. By selecting the element with the highest score in a path and removing the overlapping elements, the system with the *Focussed* strategy retrieves only focused elements. Though the systems can exclude redundancy by using the *Focussed* strategy, we can not find the non-focused elements in the result and may lose some possible benefits of XML-IR [2]. A system with the *FetchBrowse* strategy first identifies relevant documents (the fetching phase) and then identifies relevant elements within a fetched document (the browsing phase).

We considered that following problems exist in the element retrieval of XML-IR.

D.H.-L. Goh et al. (Eds.): ICADL 2007, LNCS 4822, pp. 230–240, 2007.

– *Variety of element size*
  Each element retrieved by XML-IR systems varies widely in size. An element may be large one such as root element, which is corresponding to whole document, or small one. Therefore, the cost for reading the content of a retrieved element is unknown beforehand.
– *Handling nesting elements*
  When users browse the content of nesting elements, by browsing the content of an ancestor element, users can browse the content of all its descendant elements. Though it is important to take the nestings into consideration, the *Focussed* strategy, which is the only strategy in INEX 2005 that considers about nestings, lacks flexibility because it does not retrieve elements except focused elements.

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 [3,4]. However, total output size of top-k elements is uncontrollable by simply giving an integer k. We considered that top-k search is not suited for XML-IR, and using the cost for reading the content of a retrieved element, which we call *reading effort*, instead of an integer k is better alternative to control the total output size. In addition, the search results of top-k search using the *Thorough* strategy may contain nesting elements; therefore top-k search is not suited for XML-IR also in this point.

For example, when we retrieve top-100 result elements using our system proposed in [5] with the *Thorough* strategy, The total result size of top-100 elements are varied about 30 times between the largest and the smallest for 40 queries of INEX 2005. We also observed that about 15% of top-100 total result size was overlapping content on average.

To overcome these problems, we introduced *benefit* which is the amount of gain by reading the element and *reading effort* which is the cost of reading elements, and used them in result retrieval and ranking algorithm. By considering that the *benefit* does not increase even if we read the same content repeatedly, we handle the problem of content overlapping by nestings. We supposed that users specify the threshold amount of *reading effort* they can spend in reading the result elements. The system retrieves elements that have larger *benefit* within the specified *reading effort*. The scores for result elements can be viewed as efficiency of obtaining *benefit* from the element, and we supposed the score of an element is calculated by dividing *benefit* by *reading effort*. We also propose an evaluation metrics based on *benefit* and *reading effort*.

To handle nestings in search results of XML-IR, Clarke [2] proposed to control overlapping by re-ranking the descendant and ancestor elements of the reported element. Clarke supposes each result element can be browsed with the same cost, whereas we introduced *reading effort* for the ranking algorithm and the evaluation metrics.

For evaluation metrics, we used assessments for XML test collection of INEX 2005. We compared the metrics with existing XML-IR metrics by experiments, and found there is not strong correlation between them.

2 Benefit and Reading Effort

To handle nesting elements and variety of element size, we introduced *benefit* and *reading effort*. In this section, we discuss the properties of *benefit* and *reading effort*.

2.1 Benefit

For a given query, the *benefit* of an element is the amount of gain about the query by reading the element. We describe the *benefit* of element e as $e.benefit$. Basically, it is natural to consider that the *benefit* of an element is sum of the *benefit* of the child elements. However, by reading all child elements together, the content of the child elements may complement each other; hence the *benefit* of the parent element may larger than sum of the *benefit* of the child elements. We considered that the following assumption holds on *benefit*.

Assumption 1. Property of Benefit
The *benefit* of an element is greater than or equal to the sum of the *benefit* of the child elements.

2.2 Reading Effort

The *reading effort* of an element is the amount of cost by reading the content of the element. We describe the *reading effort* of element e as $e.reading\_effort$. Note that *reading effort* does not depend on queries and can be calculated based on the element itself. Basically, it is natural to consider that the *reading effort* of an element is sum of the *reading effort* of the child elements. However, by reading all together, we can continuously read with the same context; hence the *reading effort* of the parent element may smaller than sum of the *reading effort* of the child elements. We considered that the following assumption holds on *reading effort*.

Assumption 2. Property of Reading Effort
The *reading effort* of an element is less than or equal to the sum of the *reading effort* of the child elements.

3 Ranking Method Based on Benefit and Reading Effort

XML-IR systems we propose calculate the *benefit* of elements for the given query, and rank the result elements in order of efficiency to obtain *benefit*. We supposed the score is calculated by dividing *benefit* by *reading effort*, and it corresponds to the efficiency to obtain *benefit*. We describe the score of element e as $e.score$. We also supposed that users specify the threshold amount of *reading effort*, and the system retrieves elements that have larger *benefit* within the specified *reading effort*.

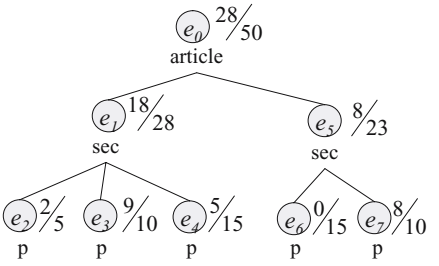

Fig. 1. An example of calculated *benefit* and *reading effort*

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 do not assume any concrete calculation formula for *benefit* and *reading effort* in Figure 1, however the values meet Assumption 1 and Assumption 2.

When a system calculate *benefit* and *reading effort* for a query as Figure 1, if a user specifies the threshold of *reading effort* to 15, the element set that maximize *benefit* is $\{e_3, e_2\}$, whereas if 20 is specified, the element set that maximize *benefit* is $\{e_3, e_7\}$. The problem of maximizing *benefit* is a variant of knapsack problems that has restriction of nestings. However, the system in the running example that maximize *benefit* does not output the content of e_2 when 20 is specified as the threshold of *reading effort*, though the content of e_2 is output when 15 is specified. Therefore, a user who specifies the threshold of *reading effort* to 20 can not obtain information from e_2 though she/he pays more *reading effort* than a user who can obtain information from e_2 by specifying 15. To avoid such situation, we considered it is important that systems have the property of the search result continuity. The search result continuity is the property defined as following.

Definition 1. *Search result continuity*
When we describe the result element set for the threshold r of reading effort as $E^r = \{e_1^r, e_2^r, ..., e_n^r\}$, and the result element set for the threshold r' as $E^{r'} = \{e_1^{r'}, e_2^{r'}, ..., e_m^{r'}\}$, the system has the property of the search result continuity if the following holds for any r and r'. The function ancestor-or-self (e) returns element set that consist of ancestor elements of e and e itself.

$$\textit{if } r \leq r' \textit{ then } \forall\ e \in E^r, \exists\ e' \in E^{r'} \textit{ s.t. } e' \in \textit{ancestor-or-self}\ (e) \qquad \square$$

In other words, the content of element set for *reading effort* r must be contained in the content of element set for *reading effort* r' if we increase the threshold value of *reading effort* from r to r'.

In addition, when we consider about ranking, we need to take the overlapping of content by nestings into account. Two patterns are possible for overlapping.

1. **Contain:** The content of a result element contains the content of upper ranked result element.
2. **Contained:** The content of a result element is contained by the content of upper ranked result element.

We think practical systems must hold search result continuity, and the system greedy retrieve result elements form higher scored elements considering nestings. The system first retrieves result elements by the *Thorough* strategy using the score based on *benefit* and *reading effort*. Then, while removing the overlapping of content by nestings, the system retrieves result elements from higher scored elements up to the threshold value of *reading effort* user input.

For removing the overlapping of content by nestings, if the pattern of **Contain** occurs the system removes the descendant elements that are already reported from the result list, and add the result to result list. If the pattern of **Contained** occurs the system simply skips the result element because the content of the element is already obtained.

When a result element e is retrieved, the *benefit* and *reading effort* of the ancestor element e_a is affected by e. In the case that the system retrieves e_a after e, the increment of *benefit* by e_a is $e_a.benefit - e.benefit$, and the increment of *reading effort* to obtain the increment of *benefit* by e_a is $e_a.reading\_effort - e.reading\_effort$.

When we suppose that users specify the threshold value of *reading effort*, the ranking algorithm based on *benefit* and *reading effort* considering nestings is shown in Figure 2.

As an example of ranking scheme, we show the case when a user specifies the threshold of *reading effort* to 40, and *benefit* and *reading effort* are calculated like Figure 1. The system retrieves ranked result element list using the *Thorough* strategy. In this case, the list is $\{e_3(e_3.score = 9/10 = 0.9)$, $e_7(0.8)$, $e_1(0.64)$, $e_0(0.56)$, $e_2(0.4)$, $e_5(0.35)$, $e_4(0.33)\}$. This list and threshold of *reading effort* are input of the ranking algorithm, and the list is processed from the top ranked result to lower ranked results. First, e_3 is processed and added to the output list $list_{out}$. At the same time, the *benefit* and *reading effort* of the ancestor elements e_1 and e_0 are adjusted. For e_1, $e_1.benefit$ is decreased to 9 and $e_1.reading\_effort$ is decreased to 18, and thereby $e_1.score$ is set to 0.5 (9/18). For e_0, $e_0.benefit$ is decreased to 19 and $e_0.reading\_effort$ is decreased to 40, and thereby $e_0.score$ is set to 0.48 (19/40). The system reflects these adjustments, and re-rank the elements in $list_{in}$. In this case, $list_{in}$ becomes $\{e_7(0.8)$, $e_1(0.5)$, $e_0(0.48)$, $e_2(0.4)$, $e_5(0.35)$, $e_4(0.33)\}$. Then, e_7 is processed and $list_{out}$ becomes $\{e_3, e_7\}$, and $list_{in}$ becomes $\{e_1(0.5), e_2(0.4), e_0(0.37), e_4(0.33), e_5(0)\}$. Next, e_1 is processed and the system removes e_3 from $list_{out}$ because e_3 is the descendant of e_1. $list_{out}$ becomes $\{e_7, e_1\}$, and $list_{in}$ becomes $\{e_2(0.4), e_4(0.33), e_0(0.17), e_5(0)\}$. Subsequently, e_2 and e_4 are processed, however they are skipped because the ancestor element e_1 is already in $list_{out}$. Next, e_0 is processed, however if we retrieve e_0, the cumulated reading effort exceeds the specified *reading effort*, so the processing terminates and the system outputs $list_{out}$ $\{e_7, e_1\}$ as the final result.

```
Input:  list_in, // result list of Thorough
        reading_effort_t // threshold of reading effort
Output: list_out

reading_effort_c = 0 // cumulated reading effort
while ((e = top(list_in))! = null) do
  remove e from list_in
  skip = false
  remove = empty
  for (e_o in list_out) do
    if (e_o is ancestor of e) then
      skip = true
      break
    end if
    if (e_o is descendant of e) then
      add e_o to remove
    end if
  end for
  if (skip) then
    continue
  end if
  reading_effort_c+ = e.reading_effort
  if (reading_effort_c > reading_effort_t) then
    break
  end if
  for (e_d ∈ remove) do
    remove e_d from list_out
  end for
  add e to list_out
  for (e_a ∈ e.ancestors) do
    e_a.benefit− = e.benefit
    e_a.reading_effort− = e.reading_effort
    rerank e_a in list_in
  end for
end while
return list_out
```

Fig. 2. Ranking algorithm based on *benefit* and *reading effort*

4 Evaluation Metrics

For evaluating systems based on *benefit* and *reading effort*, we can compare cumulated *benefit* by the system with cumulated *benefit* by the system which knows actual *benefit* for each element for a certain threshold of *reading effort*. The system which knows actual *benefit* for each element can use the best list of *Thorough* as input, and we call this system BTIL (Best Thorough Input List) system. Implementers of XML-IR systems develop better system by guessing the *benefit* of each element close to the actual *benefit*. We supposed that we can use

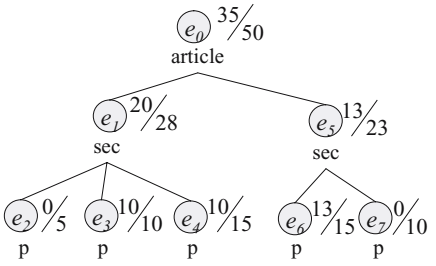

Fig. 3. Actual *benefit* and *reading effort*

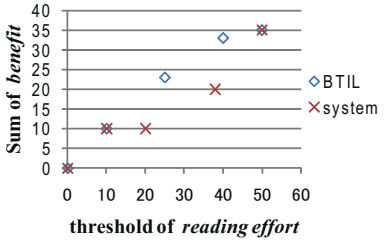

Fig. 4. b/e graph

common *reading effort* value between the BTIL system and the system to be evaluated, because *reading effort* is the value that is not depend on queries.

As an example, we explain about the case the system calculates *benefit* and *reading effort* like Figure 1, however the actual *benefit* and *reading effort* are those shown in Figure 3. In this case, for the threshold value of *reading effort* 40, the system can obtain 20 *benefit* by retrieving $\{e_7, e_1\}$, however the BTIL system can obtain 33 *benefit* by retrieving $\{e_3, e_6, e_4\}$.

We can draw a graph by plotting the cumulated *benefit* by the system and the cumulated *benefit* by the BTIL system changing the threshold of *reading effort*. We call this graph benefit/effort graph (b/e graph, for short), and use for evaluation. Figure 4 shows the b/e graph for the running example. In Figure 4, 'BTIL' is for BTIL system and 'system' is for the system to be evaluated.

In this b/e graph, for a given *reading effort* r, the obtained *benefit* is the *benefit* of the point having maximum *reading effort* under r. For example, if the threshold value of *reading effort* was 30, the obtained *benefit* for 'BTIL' is 23 and 10 for 'system'. The b/e graph enables us to intuitively understand the performance of the system compared to the BTIL system.

5 Experiments

We used test collection for XML-IR provided by INEX 2005 project [1]. The test collection consists of XML documents, queries called topics, and relevance assessments. We implemented a system using *benefit* and *reading effort*, and obtained b/e graphs for some topics of INEX 2005. In addition, we examined the correlation between the metrics based on b/e graph and the existing XML-IR metrics.

5.1 Assessments of INEX 2005

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

[1] The assessments of INEX 2006 only use Specificity.

exhaustive (PE), and Not exhaustive (NE) [2]. We converted 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 as $ex(e)$, $sp(e)$, $rsize(e)$, $size(e)$. The following formulas hold from the properties of ex, sp, $rsize$, and $size$. $e.parent$ is the parent element of e and $e.children$ is the child element set of e.

$$sp(e) = rsize(e)/size(e) \tag{1}$$

$$ex(e) \leq ex(e.parent) \tag{2}$$

$$rsize(e) = \sum_{e_i \in e.children} rsize(e_i) \tag{3}$$

$$size(e) = \sum_{e_i \in e.children} size(e_i) \tag{4}$$

5.2 Calculation of Actual Benefit and Reading Effort

We considered calculating actual *benefit* and *reading effort* from assessments of INEX 2005. We used following equations.

$$e.benefit = ex(e)^{\alpha} * rsize(e)^{\beta} \quad (\alpha \geq 0,\ \beta \geq 1) \tag{5}$$

$$e.reading\_effort = size(e)^{\gamma} \quad (0 \leq \gamma \leq 1) \tag{6}$$

Equation 5 satisfies Assumption 1 from Equation 2 and 3, and Equation 6 satisfies Assumption 2 from Equation 4. Though the assessments of INEX 2006 only use Specificity, the above equation is compatible with them by setting $\alpha = 0$.

5.3 System Implementation

The focus of system implementation is how to calculate *benefit* because we supposed that we can use common *reading effort* value, which is not depend on queries, with the BTIL system. We need the calculation formula for *benefit* that satisfies Assumption 1.

For this experiment, we used a formula for *benefit* based on $tf - ief$ [3]. When the $tf - ief$ is large, it is considered that we can obtain much information about the topic in the input query, so we can guess *benefit* is large. In addition, when multiple terms are used in the input query, we considered that *benefit* becomes larger if the element contains more terms in the query.

$$e.benefit = \frac{n}{|q|} * \sum_{t \in q} (tf * ief) \tag{7}$$

[2] Too Small (TS) is introduced for small elements, however we regard TS is equal to NE.

[3] ief stands for inverse element frequency.

$$ief = ln\frac{N+1}{ef} \tag{8}$$

where tf is the term frequency, ief is the inverse element frequency. Here, n is the number of terms occurring in both q and e, q is the input query, $|q|$ is the number of terms in q, N is the number of all elements, and ef is the number of elements that the term occurs. Though some term weighting schemes for XML documents are proposed [5,6,7], we used simple formula which satisfies Assumption 1, as we considered it is important to satisfy Assumption 1.

5.4 b/e Graph

We obtained b/e graphs of the system in Section 5.3, considering that the actual *benefit* and *reading effort* are given by the scheme in Section 5.2. As examples, we show the b/e graphs of the four topics; Topic 203, Topic 206, Topic 207, and Topic 210 of INEX 2005 in Figure 5, Figure 6, Figure 7, and Figure 8, respectively. In this experiment, we set $\alpha = 0.5$, $\beta = 1$, and $\gamma = 1$. Although the sum of *benefit* by the system we implemented is rather low compared to BTIL system especially in Topic 206, note that the performance of the system is not related to the usefulness of the proposed scheme.

We examined the comparison of the metrics based on b/e graph with existing XML-IR metrics. Using 29 Topics of INEX 2005, we calculated for each topic iMAep (interpolated Mean Average effort precision) [8], which is one of the existing XML-IR metrics using result list of *Thorough*, and iMArep (interpolated Mean Average reading effort precision), which is calculated based on b/e graph

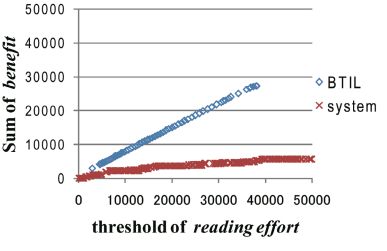

Fig. 5. b/e graph of Topic 203

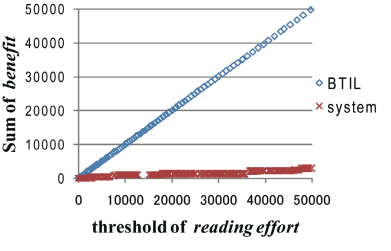

Fig. 6. b/e graph of Topic 206

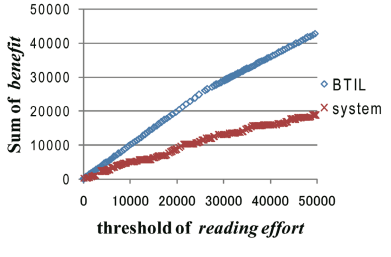

Fig. 7. b/e graph of Topic 207

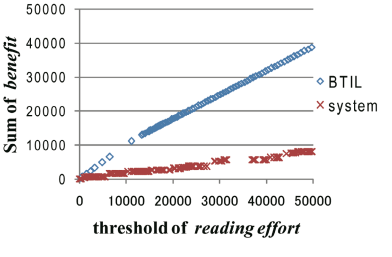

Fig. 8. b/e graph of Topic 210

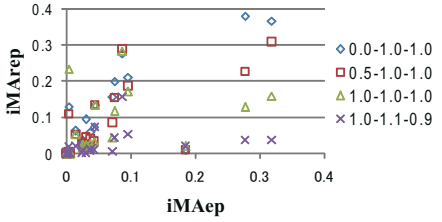

Fig. 9. Relationship between iMAep and iMArep

Table 1. Correlation coefficient between iMAep and iMArep

α-β-γ	correlation coefficient
0.0-1.0-1.0	0.81
0.5-1.0-1.0	0.75
1.0-1.0-1.0	0.48
1.0-1.1-0.9	0.39

with similar concept of iMAep. We used linear interpolation of BTIL plots on b/e graph as ideal for iMArep. iMAep is calculated based on the rank, while iMArep is calculated based on the *reading effort*. The relationship between iMAep and iMArep is shown in Figure 9. In Figure 9, we examined with four patterns of parameters α, β, and γ, and they are shown in the form of α-β-γ. Then, we examined correlation between iMAep and iMArep for one measure of effectiveness that we evaluate systems based on b/e graph. Table 1 shows the correlation coefficients of each pattern. In the case of 0.0-1.0-1.0 or 0.5-1.0-1.0, the correlation is relatively strong, however in the case of 1.0-1.0-1.0 or 1.0-1.1-0.9, we can say the correlation is weak. As the correlation of the pattern 1.0-1.1-0.9, which is considered to be close to the actual situation, is weak, we think the system can be evaluated with different measure to existing metrics by using the b/e graph based evaluation.

6 Conclusions

We introduced the concept of *benefit* and *reading effort* for XML-IR systems, and proposed the ranking algorithm and evaluation metrics based on them. The system retrieves the result elements considering efficiency and removing the overlapping of content by nestings.

Future works include introducing the concept of *switching effort*, which is the cost of switch the result item in the result list, as many results will increase the cost of browsing. Furthermore, for the XML documents created by marking up original PDF files, it is natural to show search result elements mapped on a physical page image [9], and integration with such user interface is also one of our future works. A major drawback of our current scheme is that users must specify the threshold of *reading effort*. We believe that developing user interfaces that can smoothly retrieve result elements when users change the threshold value of *reading effort* is a promising solution.

References

1. Malik, S., Kazai, G., Lalmas, M., Fuhr, N.: Overview of INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 1–15. Springer, Heidelberg (2006)

2. Clarke, C.L.A.: Controlling overlap in content-oriented XML retrieval. In: SIGIR, pp. 314–321 (2005)
3. Theobald, M., Schenkel, R., Weikum, G.: An efficient and versatile query engine for TopX search. In: VLDB, pp. 625–636 (2005)
4. Kaushik, R., Krishnamurthy, R., Naughton, J.F., Ramakrishnan, R.: On the integration of structure indexes and inverted lists. In: ACM SIGMOD, pp. 779–790 (2004)
5. Shimizu, T., Terada, N., Yoshikawa, M.: Kikori-KS: An effective and efficient keyword search system for digital libraries in XML. In: Sugimoto, S., Hunter, J., Rauber, A., Morishima, A. (eds.) ICADL 2006. LNCS, vol. 4312, pp. 390–399. Springer, Heidelberg (2006)
6. Grabs, T., Schek, H.-J.: ETH Zürich at INEX: Flexible information retrieval from XML with PowerDB-XML. In: INEX, pp. 141–148 (2002)
7. Amer-Yahia, S., Curtmola, E., Deutsch, A.: Flexible and efficient XML search with complex full-text predicates. In: ACM SIGMOD, pp. 575–586 (2006)
8. Kazai, G., Lalmas, M.: INEX 2005 evaluation measures. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 16–29. Springer, Heidelberg (2006)
9. Shimizu, T., Yoshikawa, M.: XML information retrieval considering physical page layout of logical elements. In: WebDB (2007)