Okapi based XML indexing

Wei Lu[1], Andrew Macfarlane[2*] and Fabio Venuti[2],

[1] Center for Studies of Information Resources, School of Information Management, Wuhan University, China

[2] Centre for Interactive Systems Research, Department of Information Science, City University London EC1V OHB

*corresponding author: andym@soi.city.ac.uk

Abstract

Being an important data exchange and information storage standard, XML has generated a great deal of interest and particular attention has been paid to the issue of XML indexing. Clear use cases for structured search in XML have been established. However, most of the research in the area is either based on relational database systems or specialized semi-structured data management systems. In this paper, we propose a method for XML indexing based on the Information Retrieval (IR) system Okapi. Firstly, we review the structure of inverted files and give an overview of the issues of why this indexing mechanism cannot properly support XML retrieval, using the underlying data structures of Okapi as an example. Then we explore a revised method implemented on Okapi using path indexing structures. We evaluate these index structures through the metrics of indexing run time, path search run time and space costs using the INEX and Reuters RVC1 collections. Initial results on the INEX collections show that there is a substantial overhead in space costs for the method, but this increase does not affect run time adversely. Indexing results on differing sized Reuters RVC1 sub-collections show that the increase in space costs with increasing the size of a collection is significant, but in terms of run time the increase is linear. Path search results show sub-millisecond run times, demonstrating minimal overhead for XML search. Overall, the results show the method implemented to support XML search in a traditional IR system such as Okapi is viable.

General Terms: Indexing methods

Additional keywords and phrases: Information Retrieval; XML indexing; efficiency evaluation; data structures;

1. Introduction

With the increase of information available on the Internet, the issue of managing semi-structured data has gained some attention. As a popular syntax for semi-structured data, XML is becoming more important in data exchange and information storage. Clear use cases for XML search have been

established at INEX (Trotman et al, 2007), and a need for structural elements for queries have been established by (Woodley et al, 2007) for situations where users have multiple information requests. A great deal of research has been conducted in XML indexing to support powerful, flexible and efficient XML retrieval. (Cooper et al. 2001, Gou and Chirkova, 2007) state that there are usually two ways to index XML data. One option is to store it with a Relational Database Management System (RDBMS). An example of this is (Florescu & Kossman, 1999), who map XML documents into relational tables. This method usually requires a schema for the data. If no schema exists, the data can be stored as a set of data elements and parent-child nesting relationships. Systems such as STORED (Deutchsch et al. 1999) and XISS/R (Harding et al. 2003) use this method. Another option is to build a specialized data manager for XML storage and indexing. Projects such as Lore (McHugh et al. 1997) and industrial products such as Tamino and MarkLogic take this approach. This type of system has a great deal more flexibility than the RDBMS approach, but without having the benefit for users of the extensive knowledge gained with relational systems over the years. Wei & Da-xin (2005) put forward a method of provIDing access to XML documents using a hybrID method with both database and IR techniques utilized, but are focused on serving both database and IR queries.

XML indexing must support both **path** and **value** retrieval i.e., structural and content components of XML documents. The **path** and **value** terms are defined formally as follows. XML documents can be viewed as a tree, with a **path** describing the sequence of nodes from the document root to a specific element. The **path** consists of a sequence of path steps, where each step corresponds to an element (Fuhr and Govert, 2002). Examples of a **path** in Fig. 1 are /newsitem, /newsitem/title, /newsitem/text and /newsitems/text/p etc. **Value** in this context means the content of XML documents but not the element or attributes names i.e., the text. The difference between the two methods is that **path retrieval** permits users to search specified paths or elements, while **value retrieval** permits users to search the text content of XML documents. Up to now, most of the research conducted on XML indexing was centred on path retrieval. Many index methods reported in the literature, including (Chung et al. 2002; Kaushik et at. 2003; Goldman & WIDom, 1997; Milo & Suciu, 1999), do not support value indexes (Wang et al. 2003). Some systems such as HYREX do now support value centric retrieval (Fuhr & Großjohann, 2004). There has also been a variety of indexing methods used in the INEX program recently. The XFIRM system uses a relevance propagation method to answer 'content only' (CO) and 'content and structure' CAS queries (Sauvagnat et al, 2006). Geva (2005) proposed a Microsoft Access based XML Retrieval System, which also forms the basis of the indexing structures and the kernel for the system B3-SDR (van Zwol, 2006). Fujimoto et al. (2006) developed an XML information retrieval system by using XRel, an XML database system on relational databases. Theobald et al. (2006) propose a threshold algorithm XML retrieval system for participating in INEX 2005. Some systems like EXTIRP (Lehtonen, 2006) divIDe the XML document collection into disjoint fragments and then naturally treated the fragments as traditional documents

which are independent of each other. SIRIUS, a lightweight indexing and search engine is also document oriented (Popovici et al, 2006).

Most IR systems are free text retrieval systems, which in general only support value retrieval. Over many years, these systems have played an important role in encouraging the development of IR research, particularly through such initiatives as TREC. The retrieval models embedded in them are sophisticated and we believe that they could be useful for XML value centric retrieval. This leads to a question: can traditional IR systems be modified in order to handle full XML retrieval i.e., both path and value search? In this paper, we discuss how to implement XML indexing by extending the capabilities of inverted files, in order to manage XML collections while still maintaining backward compatibility (by this we mean the ability to service **value** only retrieval if required).

The difference between XML retrieval and traditional IR is that the former requires retrieval on the element level as well as the document level. This means that both value indexes and element indexes are required. The problem then is to combine element indexes with traditional IR value indexes. In section 2 we review inverted file structures using Okapi as an example, and show why this structure is inadequate for XML retrieval. We present an indexing method that supports both value centric and data centric XML retrieval in section 3. In section 4, we evaluate our method by utilizing the measures of indexing time and size of index. We give a conclusion and outline some further work to be done at the end.

2. Inverted file data structures

There are many indexing structures which can be used to support text searching including PAT trees (Gonnet et al, 1992), but inverted files have long been recognised as being the best technology for this purpose (Harman et al, 1992; Zobel and Moffat 2006). In broad terms, this is because a set of 'postings' – documents which contain information on a particular word - are stored contiguously on disk, which facilitates fast disk access. Inverted files have a bewildering variety of different forms, but can be classed under two main formats: document level and word level (Zobel and Moffat 2006). These two formats are distinguished in the word or position data which is held in word level formats in order to support proximity operations of different types or use of phrases in queries such as 'to be or not to be'. An example of word level index data structures is Okapi inverted files (Jones et al. 1997), which have the following structure:

- The 'Primary Index' file stores the number of the block in the secondary index, which contains a keyword being searched for.
- The 'Secondary Index' file, and Dictionary file, contains blocks of keywords which occur in the collection. Each record in a block contains information on the keyword and a pointer to

the first posting for that keyword in the Postings file.

- The ‘Postings File’ contains a record for every occurrence of a term in the collection and records the term frequency and position list for that term.

Each element of the postings file has the following structure:

<tf><recnum>(<pos>).

The <tf> field contains the within-document term frequency, which has a maximum value of 16383. The <recnum> field is an unsigned value containing the internal record number (IRN) of the document. The <pos> field is variable in size and contains 32-bit record structures that store information on within-document positional information. This record structure contains five elements (see Table 1):

Table 1: Position structure used in Okapi

Field	Description
f	Field number
s	"Sentence" number within field
t	"Token" number within sentence
nt	Number of tokens making up this index term
sw	Number of stop words preceding this index term

The information recorded in this structure is used to support operations such as passage retrieval and proximity searching. However, without alteration it is unable to support the kinds of searches that are required for XML element retrieval. We illustrate this problem by using a prototype record of an XML collection in Fig. 1, taken from the Reuters RCV1 collection (Lewis et al. 2004).

```
<newsitem itemID="4929" date="1996-08-20" xml:lang="en">    <title>...</title>
<headline>...</headline>    <dateline>...</dateline>    <text>
        <p>...</p>
        <p>...</p>
  </text>
  <copyright>...</copyright>
  <metadata>
        <codes class="bip:countries:1.0">
                <code code="AUST">...</code>
        </codes>
      <dc element="dc.date.created" value="1996-08-20"/>
      <dc element="dc.publisher" value="Reuters Holdings Plc"/>
      <dc element="dc.date.published" value="1996-08-20"/>
  </metadata>
</newsitem>
```

Fig. 1: Prototype of XML record

Traditional inverted files using word level indexes (such as the Okapi example above) assume a linear sequence of elements such as Book, Chapter, Paragraph and Sentence (Zobel and Moffat 2006), which

are contiguous and non-overlapping. However they cannot represent the complex hierarchical structure of XML documents (such as those in Fig. 1), which for example may allow more complex structures such as associating titles with say Chapters as well as Books. We can use the field number in the Okapi position record for any element, but cannot record what its relation is to other elements in the hierarchy (a pathway is needed). A further problem is the IDentification of the element to retrieve – an important part of structured XML document retrieval. In Fig. 1 for example, the element "dc" is repeated several times with different attributes: there is no way for a word level index to recognize which element to address (the unrevised data structure is only able to store the offset of one element). The result is that only the last element of the sequence is consIDered, that is, following the example, <medatada><dc element> will have value 'dc.date.published' and <metadata><dc value> will have value '1996-08-20'. Word level indexes such as those used for Okapi will therefore not support full XML retrieval.

3. Indexing method to support path and value retrieval

In this section we propose an indexing structure which is able to support full XML document retrieval for both **value** centric and **path** centric cases, which essentially is an augmented inverted file. This gives us the advantages of this technology (i.e., fast searching), but also gives us the ability to extend the type of search we are able to service (see Section 2). Most of the Okapi search models are compatible with XML article level retrieval and passage retrieval models also could be modified for element level retrieval. The problem could be resolved by merging these repeated elements into one single element, thus altering the structure of the original XML document, but this is not a desirable solution as it cannot support real element retrieval.

Supporting both **value** centric and **path** centric retrieval means that consIDeration of both the value and structural information of XML documents is essential. The index data structure must therefore be able to record XML structural data, as well as value information. Being a free text retrieval system with a word level inverted list, Okapi can support XML value indexing, but does not support path indexing. We therefore have developed a comprehensive method to implement XML indexing based on Okapi like structures. Our method is divIDed into 2 stages: firstly, path indexing is executed; secondly, value indexing is performed based on the path index information.

3.1. Path indexing

(Fuhr and Govert, 2002) assert that **"**in order to process queries referring to the logical structure of documents" (please refer to Fig. 1), "XML query languages must support the following four types of conditions":

- Element names: ability to specify element name in search e.g., from Fig. 1, restrict the 'dc element' to the value 'dc.publisher'.
- Element index: ability to search on elements e.g., from Fig. 1, the 'headline' (field search).
- Ancestor/descendant: ability to use the hierarchical structure of the documents for search, e.g., from Fig. 1, find the 'metadata' then 'dc element'
- Preceding/following: ability to use the linear sequence of the document for search, e.g., from Fig. 1, 'headline' then 'text'.

All of this information must be contained in path indexes. There has been a large body of research completed on XML path indexing (Cooper et al. 2001; Deutsch et al. 1999; Harding et al. 2003; McHugh et al. 1997; Chung et al. 2002; Kaushik et al. 2002; Milo and Suciu, 1999; Wang et al. 2003). In this paper, we propose a pre-order B+ trees path index method which is similar to ViST (Wang et al. 2003) and XISS/R (Harding et al. 2003) but with a revised index structure. Unlike ViST and XISS which use a RDBMS to store path information, we show how a path index manager can be created by referencing an inverted file index structures. We use Okapi's free text structures to illustrate this process, but it can be easily adapted to other types of word level indexes.

3.2. Path index structures and algorithm

There are 3 main path index files: the *Path file*, the *Path position* file and the *Path instance offset file*. The detail structures of these files are shown in Fig. 2.

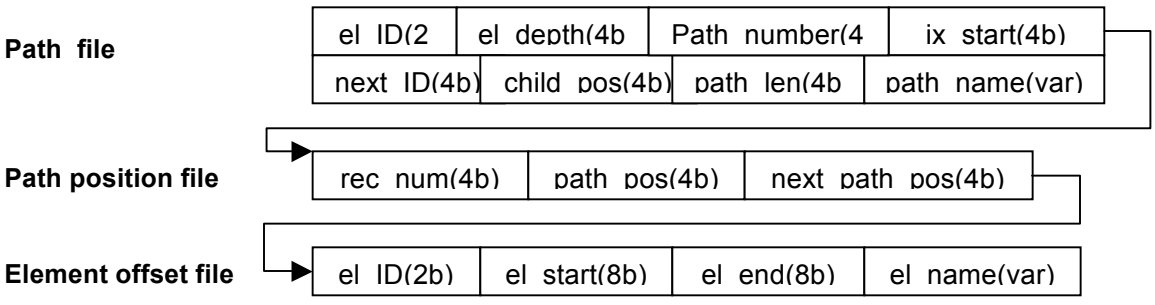


Figure 2. Structure of main path index files

where:

- *Path file* stores the path ID information (see Table 2). For each different path, a unique integer

ID is given by its occurrence order in the collection, together with the path name and path depth. In this file, **ix_start** and **ix_len** are the start offset and length of the path instance information in *Path position file*. The **next_pos** and child_pos fields point to the position of the next path in the same level and its first child, respectively, for context path positioning. Simple examples of path name from Fig. 1 are /newsitem, /newsitem/title and /newsitem/text/p, etc. This file is sorted by **path_name** in ascending order to ensure that all children paths are behind their parent path. This sequence can improve the path retrieval speed significantly by using Binary Search method for a specified path.

- *Path position file* stores path instances' position information in the *Path instance offset file* (see Table 3). In this file, **path_pos** points to the position where the path instance is in the *Path instance offset file*. All the **path_pos** for a specified path are grouped together for improving search speed. The first "0xffffffff" means the end of the path instances' occurrences in a record while the second one means the end of the path instances' occurrences in the document collection. This file is similar to a postings file described above.
- *Path instance offset file* stores path instances' position information in the XML collection (see Table 4). In this file, **path_ID** is the same as that in *Path file*, **instance_start** and **instance_end** point to the path instance's start and end positions in the XML collection, and **path_seq** is path instance's detailed information which contains element index information. For example, given an path name /article/chapter/section/p, an example of its instance is article(1)/chapter(2)/section(3)/p(2) which represents paragraph 2 in section 3, chapter 2. Accordingly, the **instance_seq** for this path is "1 2 3 2". For each record, *Path instance offset file* stores elements in pre-order traversal B+ trees which benefits both the search and value indexing speed.

We give a practical example of how data is stored in the above path index files in order to facilitate understanding. Suppose that a prototype record of an XML collection is like the one shown in Fig. 1, then see the following tables 2, 3 and 4 for the data in these files (in the example *addr* is the record start position in the path index files).

Table 2: Example data for Path file

Path address:	path_ID	depth	path_number	ix_start	ix_len	next_pos	child_pos	path_len	path_name
ID_addr1 :	1	1	1	ix_addr1	…	0xffffffff	ID_addr2	9	/newsitem
ID_addr2 :	7	2	1	ix_addr2	…	Id_addr3	0xffffffff	19	/newsitem/copyright

ID_addr3 :	4	2	1	ix_addr3	…	Id_addr4	0xffffffff	18	/newsitem/dateline
ID_addr4 :	3	2	1	ix_addr4	…	Id_addr5	0xffffffff	18	/newsitem/headline
…	…								
ID_addr8 :	11	3	3	ix_addr8	…	0xffffffff	0xffffffff	21	/newsitem/metadata/dc
…	…								

Table 3: Example data for Path position file

Path Position	rec_num	path_pos	rec_end_tag	path_end_tag
ix_addr1:	1	pi_addr1	0xffffffff	0xffffffff
ix_addr2:	1	pi_addr8	0xffffffff	0xffffffff
ix_addr3:	1	pi_addr4	0xffffffff	0xffffffff
ix_addr4:	1	pi_addr3	0xffffffff	0xffffffff
…	…	…	…	…
ix_addr8:	1	pi_addr11		
		pi_addr12		
		pi_addr13	0xffffffff	0xffffffff
ix_addr9:	…	…	…	…

Table 4: Example data for Path instance offset file

Offset address	path_ID	instance_start	instance_end	instance_seq
pi_addr1:	1	0	456	1
pi_addr2:	2	...	...	1 1
pi_addr3:	3	...	...	1 1
pi_addr4:	4	...	...	1 1
…	…			
pi_addr11:	11	...	...	1 1 1
pi_addr12:	11	...	...	1 1 2

pi_addr13:	11	...	...	1 1 3
...	...			

Taking path "/newsitem" as an example, it occurs at the beginning of the collection, so the **path_ID** is set to 1. The **path_depth** is 1 and there is only one **path_number** for this XML collection. Being a root path, it has no **next_pos** (next path position in the same level) in the *Path* file and its first child path is "/newsitem/copyright". The **ix_start** fields points to ix_addr1 in the *Path position* file. As there is only one path instance for "/newsitem", ix_addr1 in the *Path position* file ends directly with 0xffffffff0xffffffff and its **path_pos** points to pi_addr1 in the *Path instance offset* file. We can then locate the path instance's name /newsitem(1) and its corresponding offset information in the original collection. Path "/newsitem/metadata/dc" is the 11th occurring path in the collection and access to its values are different because it has three instances in the collection. So in the *Path position* file, the record number is omitted for the latter two instances because they have the same record number. The instance_seq in the *Path instance offset* file are set to "1 1 1", "1 1 2" and "1 1 3" respectively.

Obviously, for a path centric search, the Binary Search method could be used to traverse the *Path* file for an absolute path such as "/newsitem/metadata/dc". But for a vague join search such as "newsitem//dc" or "//metadata//dc" where multiple paths exist, all the paths in the *Path* file have to be searched. This is very time consuming. To solve this problem, another two index files, *Element* file and *Element position* file, are proposed to create an index on the elements for all paths in the *Path file*. The structure of these two files are shown in Fig. 3.

Path file:

el_ID	path_name	ix_start
1	/newsitem	ix_addr1
7	/newsitem/copyright	ix_addr7
4	/newsitem/dateline	ix_addr4
3	/newsitem/headline	ix_addr3
8	/newsitem/metadata	ix_addr8
…	…	…
11	/newsitem/metadata/dc	ix_addr11
…	…	…

Path position file:

path position	path_pos	next_path_pos
ix_addr1:	el_addr1	0xffffffff
ix_addr2:	el_addr2	0xffffffff
ix_addr3:	el_addr3	0xffffffff
ix_addr4:	el_addr4	0xffffffff
ix_addr5:	el_addr5	0xffffffff
…	…	…
ix_addr11:	el_addr11	ix_addr12
ix_addr12:	el_addr12	ix_addr13
ix_addr13:	el_addr13	0xffffffff
ix_addr14:	el_addr14	0xffffffff

Element offset file:

element offset	el_ID	el_name
el_addr1:	1	/newsitem(1)/title(1)
el_addr2:	2	/newsitem(1)/headline(1)
el_addr3:	3	/newsitem(1)/dateline(1)
el_addr4:	4	/newsitem(1)/dateline(1)
el_addr5:	5	/newsitem(1)/text(1)
…	…	…
el_addr11:	11	/newsitem(1)/metadata(1)/dc(1)
el_addr12:	11	/newsitem(1)/metadata(1)/dc(2)
el_addr13:	11	/newsitem(1)/metadata(1)/dc(3)
…	…	…

Fig. 3. Further path index files

where:

Element file stores all the unique element information. For each different element, a unique integer ID is given by its occurrence order in the collection as that for path in the *Path file*. In this file, **elem_start**, similar to **ix_start** in the *Path file*, is the start offset of the element instance information in the *Element position file* and elem_num is the total number of element instances or occurrences in the **path_name** of the *Path file*. Similar to *Path file*, this file is sorted by **elem_name** in ascending order. This sequence can improve the element retrieval speed significantly by using the binary search method for a specified element.

Element position file stores element instances' occurrence information in the **path_name** of the *Path file*. In this file, **path_ID** and **elem_depth** tell the ID and depth where the element occurs in the **path_name**. For example, the value of **path_ID** and **elem_depth** for element "dc" in path "/newsitem/metadata/dc" is 11 (see table 2) and 3 respectively.

Thus, for a vague join path search such as "newsitem//dc", we could easily split this path into two elements "newsitem" and "dc". And for each element, we could obtain a result path set where the element occurs in by using the above two element index files. Further, the integer value elem_depth could be used for the join of these two elements for the final path result set.

We can avoid a vague join search by using these two files, firstly, traversing all paths in the *Path file* and secondly converting the join operation into a number comparision by using elem_depth, which could improve the search speed. The path centric search evaluation is provIDed in section 4.

Fig. 4 shows the path indexing algorithm.

```
D→Document Collections, R→Record, P→Path, E→Element, B→Temp Buffer

For each R in D do
  Read all E to buffer B
  For each E in B do
    If E is a new element Then
      Give E an incremental Integer ID
      Add E to buffer B for Element file
    End
    Add E to buffer B for Element position file
  End
  Generate P by using E
  For each P in B do
    If N is a new path Then
      Give P an incremental Integer ID
      Add P to buffer B for Path file
    End
    Add P to buffer B for Path position file
    Add P instance to Path instance offset file
  End
  Sort buffer B in ascending order by path name
  Store all P in the buffer B to Path file
  Store all E in the buffer B to Element file
  Group and sort all element instances and store them to Element position file
  Group and sort all temp Path position file and merge to final Path position file
End
```

Fig. 4: Path indexing algorithm

3.3. Value indexing

Word level inverted files can easily be used to support XML value indexing, e.g., the Okapi data structures outlined in section 2 above. However, this structure cannot record which element a given term belongs to. Extra information must therefore be recorded in order to combine value indexes with path indexes. There are a number of ways to do this. In our case, we modified the position structure described in table by adding a new 32 bit field 'p' which represents within-document offset information. This strategy is at the cost of doubling the size of the position records in the Postings file.

Alterations to the indexing algorithm are minor. The only difference is that the term's position information is checked when indexing and corresponding element information (the path instance position information in the *Path instance offset file*) is stored in 'p' together with the term's other position information. The address of the path in the *Path file* also can be stored in 'p' instead of the element position information. This is particularly efficient for those users who require the path name

only, and do not need access to the *path position file*.

When doing value centric retrieval, obtaining path instances of documents is a straightforward process. The result sets for the query terms are retrieved and the data recorded in the 'p' position structure, which point to the path instance position in the *Path instance offset file*, is used to obtain the corresponding path instance's offset information in the source collection. The paths required by the search are then retrieved for the user. If path is specified in the query, then the corresponding path's **path_ID** or **path_ID** set are retrieved from the *Path file*. Using the **path_ID** or **path_ID** set, the retrieved paths can be filtered because **path_ID** is also recorded in the *Path instance offset file*. Fig. 5 shows both the path and value search process. Details of how these structures support ranking through the BM25F function can be found in (Lu et al, 2006; Lu et al, 2007).

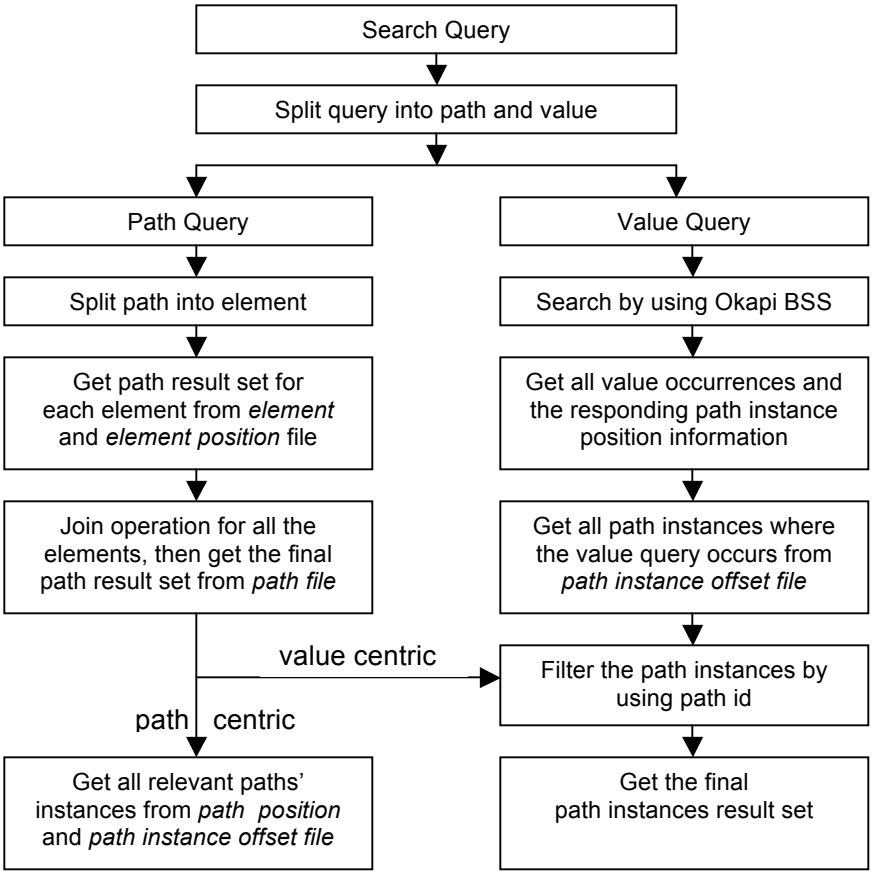


Fig. 5: Both path and value search process

4. Evaluation

We implemented our revised indexing method in C++. The operating system used for the experiments is Linux 9.0, on a dual i686 processor with 1GB of main memory. There are a number of different approaches for characterizing efficiency: we use the indexing time, size of index and path search speed measures. The comparisons for the evaluation of the indexing time and size of index are done using ordinary value indexing runs as against runs with path indexing and value indexing. We compare runs on collections of one size to measure performance on static collections and on various sizes to examine the issue of scalability. The data collections chosen also have different element complexity levels, as the tree structures XML hierarchies may vary consIDerably. The purpose of these experiments is to quantify the difference in the chosen metrics, demonstrating the viability of the algorithm and data structures described above. We describe the data sets used for experimentation in section 4.1 and analyse results of indexing runs in section 4.2.

4.1 Data Sets

We selected four data sets for our experiment: INEX 1.4 (Malik et al. 2005), INEX 1.6 (Malik et al. 2006), Shakespeare's Plays (Bosak 2006) and the Reuters RCV1 collection (Lewis et al. 2004):

INEX 1.4: This data set was used for the INEX 2004 evaluation and contains IEEE Computer Society articles dating from 1995 to 2002.

INEX 1.6: This data set was used for the INEX 2005 evaluation and contains IEEE Computer Society articles dating from 1995 to 2004.

Shakespeare's Plays: This data set contains the 37 plays of Shakespeare marked up in XML format.

Reuters RCV1 collection: A set of newswire articles from Reuters, split into subsets to test scalability.

Tables 5 and 6 give more details on the various statistics on these collections.

Table 5: Benchmark parameters (INEX and Shakespeare collections)

Data sets	**INEX 1.4**	**INEX 1.6**	**Shakespeare Works**
Size of Data(MB)	494	705	9.99
# of elements	8239873	11411135	179689
# of attributes	2204688	4669699	179689
# of Records	12107	16819	37

Avg. Path Level	8	8	5

Table 6: Benchmark parameters (Reuters RCV1 subsets)

Data sets	Reuters 1	Reuters 2	Reuters3	Reuters 4	Reuters 5	Reuters 6
Size of Data(MB)	250	500	750	1000	1500	2000
# of elements	3491446	6937705	10362907	13802112	20703163	26998953
# of attributes	3659853	7334929	10954155	14590239	21832916	28445398
# of Records	89114	178121	267052	355060	531744	692874
Avg. Path Level	6	6	6	6	6	6

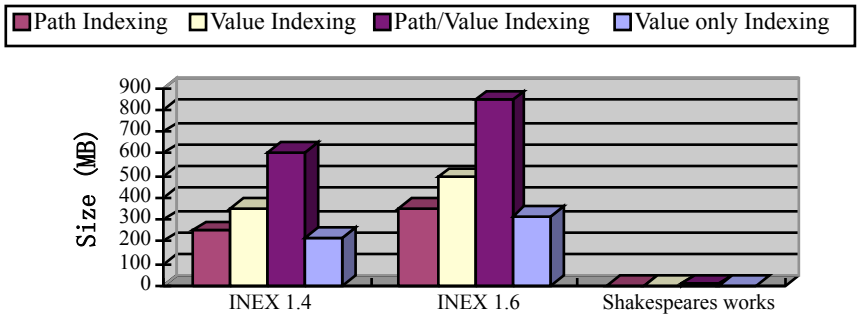
4.2 Experimental results

Fig. 6 shows a comparison of the index size among path indexing, value indexing, path/value indexing and value only indexing (here value indexing means the revised okapi text indexing, while value only indexing is the traditional okapi text indexing. Path/value indexing consists of both path indexing and value indexing). From this data, we can see that the size of the path/value index is a little larger than the XML original data size and more than two times of the value only index. For example, the INEX 1.4 source collection size is 494 MB, while the index size of the path, value, path/value and value only method are 252MB, 352MB, 604MB and 223MB respectively. The recorded index size of the value only method is less than half of the original data size and the total index size, while the path/value index is nearly 1.2 times of the original data size. This means the index size of the path/value method is 2.7 times of that of the value only index. Even the value index size of the path/value method, which is 352MB for INEX 1.4, is much larger than that of the value only method. The main reason for this is that we add 32 bytes to the position structure which nearly doubles the size of the value index size and we create an *Path position file* for locating each path instance in the *Path instance offset file*.

Fig. 7 shows the comparison of indexing run time among path indexing, value indexing, path/value indexing and value only indexing. From this figure, we can see that the path indexing is efficient and most of the path/value indexing time spent on value indexing, which is largely determined by the Okapi indexing system. For example, the path, value, path/value, and value only indexing run time are 67, 681, 748, and 604 seconds respectively. Though path position information is consIDered in value indexing, the indexing run time only is slightly over that of the value only indexing method. The total path/value indexing run time is increased only about 23% than that of value only indexing method.

For small collections such as the Shakespeare Works, the indexing completes very quickly.

Fig. 6: Comparison of index size



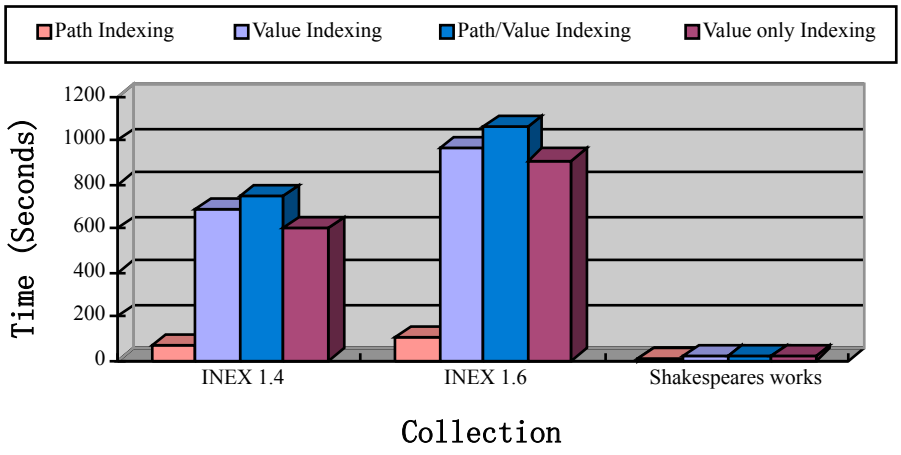


Fig. 7: Comparison of indexing run times

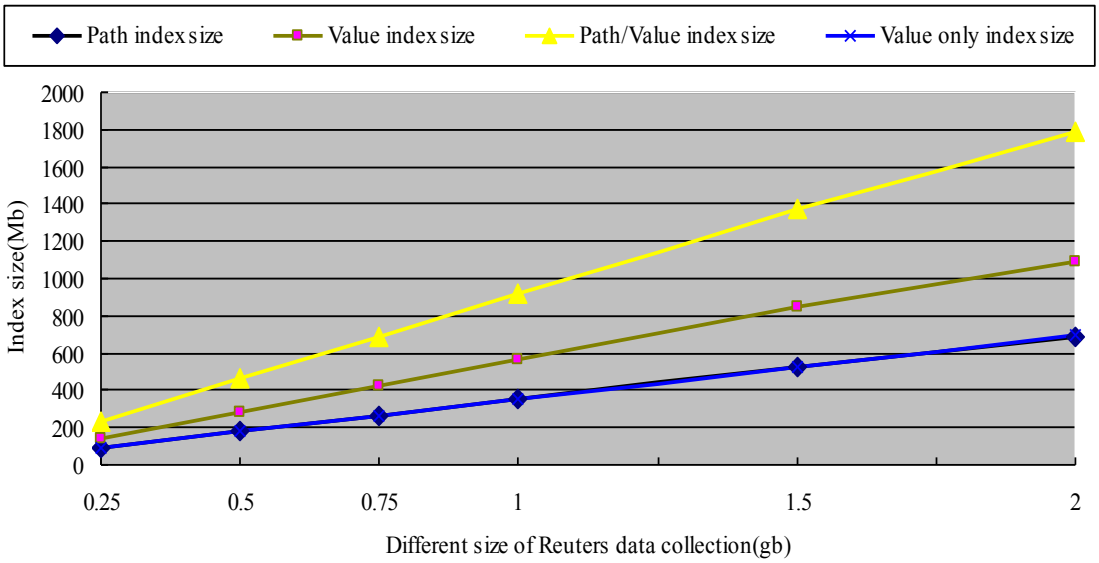


Fig. 8: Scalability of the Indexing using subsets of the Reuters RCV1 collection's (size).

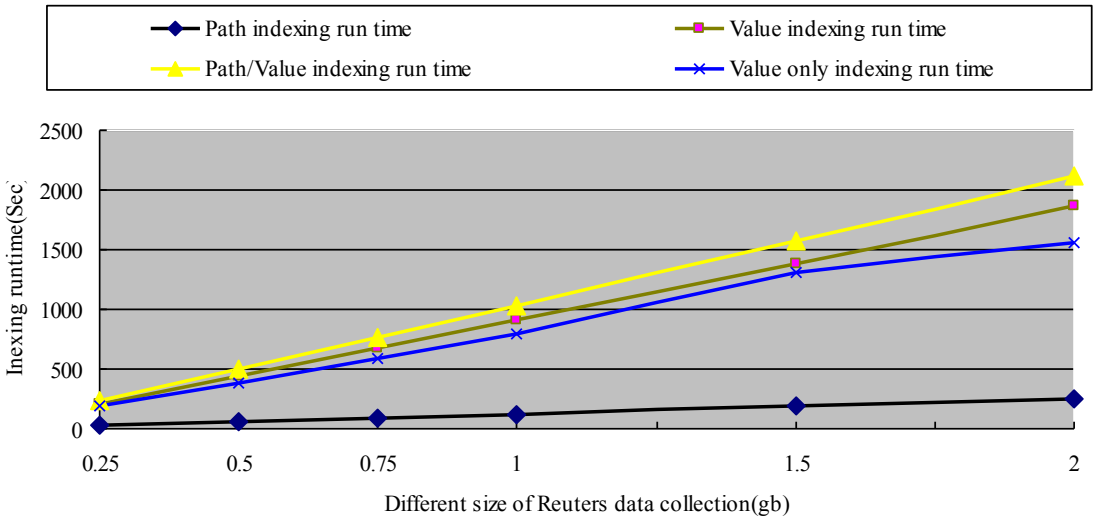


Fig. 9: Scalability of the Indexing using subsets of the Reuters RCV1 collection's (run time).

To investigate the scalability of indexing with the growth of the data collections size, we measured both indexing size and run times using various subsets of the Reuters RCV1 collection, i.e., 0.25GB, 0.5GB, 0.75GB, 1GB, 1.5GB and 2GB. The results of these runs are shown in Fig. 8 and Fig. 9. From these two figures we can see that both the index size and indexing run time increase linearly with the growth of the data collection's size. Similar to the above experiment, the revised total index size is much larger than that of the original method (about three times) but a little smaller than the original data collection's size. Results also show that the increase in indexing run time is not excessive as the path indexing is very fast and the revised value indexing run time is similar to that of the value only method. Comparing the results to the INEX collection (INEX 1.4 and INEX 1.6) of the same size, Reuters RCV1 collection has a smaller index size, and while indexing the run time is faster. This is because there are only about 80 nodes in each document in the Reuters RCV1 collection while more than 1000 nodes are contained in documents from the INEX collection. The results from Figs. 8 and 9 show that growth of indexing size and run time is linear with the size of the target source collection, demonstrating the practicality of the approach.

Table 7 shows some examples from a path search experiment. Both absolute path and vague path are tested on our indexing structures. The test collection is INEX 1.4, and three absolute paths and three vague paths are randomly selected from the *Path file*. Our search aim is to get all the relevant path instance position information and display (or not display) the top 20 results. For the absolute path, the *Path file are used* directly, while for the vague path, the *Element file and Element position file are used for join search*.. From the table, we can see that the search time is relevant to the occurrences of the element in the collection. For example, both "bdy" and "sec" occur often in the collection that any join operation is therefore more expensive. The average search time on random selected 50 queries for absolute path and vague path respectively are shown in Table 8. Results show that the path only search is quite efficient, and also the path index structure is quite flexible in supporting any kind of queries.

Table 7: Path search experiment on INEX 1.4 data set

Path type	**Path query**	**# of relevant path**	**# of relevant path instance**	**Cost time display (millisecond)**	**Cost time no display (millisecond)**
Absolute	/article/bdy/sec	1	65407	7	0.09

path	/article/fm/abs/p	1	8095	6	0.09
	/article/bm/vt/p/it/b	1	229	7	0.1
Vague path	//bdy//p	454	674285	7	0.42
	//fm//p3	2	15	7	0.1
	//sec//li//it	598	116607	6	0.4

Table 7: 50 paths search experiment on INEX 1.4 data set

Path type	**Number of path query**	**Avg. cost time display (millisecond)**	**Avg. cost time no display (millisecond)**
Absolute path	50	4	0.06
Vague path	50	4	0.31

5. Conclusions and further work

We have developed a method for XML path and value indexing and demonstrated a practical way to combine them with a traditional text retrieval system, namely Okapi. Much of the system's benefits are inherited both for value indexing and XML retrieval. Our system performed well, when participating in the INEX evaluation for the first time in 2005 (Lu et al. 2006), and we have continued to build on this work using the index structures described in this paper (Lu et al. 2007; Robertson et al. 2006). The results on index size and indexing run time measures show that while index size is increased significantly, this is not reflected in an indexing run time increase. In any case, the results show that the new method for indexing is viable as disk space is cheap and indexing time is secondary to search time for retrieval systems. By sacrificing index speed and storage space, we are able to service other types of querying, not previously available with the Okapi system. Similar systems, using word level indexes, would be able to implement these IDeas easily. Our initial path search experiments show impressive results, particularly for absolute path runs – all runs show sub-millisecond run times. The overhead for servicing these types of query are minimal.

However, further work must be completed in order to provIDe full XML search facilities using the path/value indexing method. A more powerful XML query parsing and display system needs to be developed based on Okapi's BSS system. We have already developed a simple interface for parsing CO (Content Only) queries, but our system cannot support structured query parsing as yet. As XML requires element level retrieval, a method to display relevant elements based on Okapi still needs to be investigated. Even regarding indexing, some problems such as element type and attribute structures etc. still need to be resolved. Our indexing system does not consIDer an XML element's data type,

e.g., numeric, date, integer, etc. All the values of elements are treated as strings or text, which we believe should be upgraded to improve retrieval efficiency. Furthermore, attributes are ignored both by path indexing and value indexing in our current methods. Whether to treat an attribute as a special element or propose a specific structure to index such data is an open question. We will investigate these issues in further research.

Acknowledgements

This research was partially funded by Microsoft Research CambrIDge in the project "Improving tools for investigating linguistic and probabilistic models in IR: an XML indexer for Okapi". Thanks go to the China Scholarship Council (CSC) and Wuhan University for funding the first author's visit to City University, London in order to conduct this research.

References

CHUNG, C, MIN, J. & SHIM, K. APEX: An adaptive path index for XML data. *In: Franklin, M.J. Moon, B. & Ailamaki, A, (Eds.), Proceedings of ACM SIGMOD 2002 Conference, Madison Wisconsin, USA,* (2002), 109-120.

COOPER, B.F., SAMPLEM N., FRANKLINM M.J., HJALTASON, G.R, & SHADMON, M. A Fast Index for Semistructured Data. *In: Apers, M.G. Atzeni, P., Ceri, S. Paraboschi, S. Ramamohanaroa, K. & Snodgras, R..T. (Eds.), Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy*, (2001), Morgan Kaufmann, 341-350.

DEUTSCH, A., FERNANDEZ, M. & SUCIU, D. Storing semistructured data with STORED. *In: Delis, A., Faloutsos, C. & Chandeharizadeh, S. (Eds.), Proceedings of ACM SIGMOD Conference 1999, Philadelphia, Pennsylvania, USA*, (1999), 431-442.

FLORESCU, D. & KOSSMANN, D. Storing and Querying XML Data using an RDMBS, IEEE Data Engineering Bulletin, Vol. 22, No. 3., (1999), 27-34.

FUHR, N. & GOVERT, N. Index Compression vs. Retrieval Time of Inverted Files for XML Documents. *In Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA,* (2002), 662-664.

FUHR, N. & GROßJOHANN, K. XIRQL: An XML Query Language Based on Information Retrieval Concepts, *ACM Transactions on Information Systems*, Vol. 22, No. 2., (2004). 313-356.

FUJIMOTO, K. SHIMIZU, T. TERADA. N. HATANO, K, SUZUKI, Y, AMAGASA, T. KINUTANI, H. & YOSHIKAWA, Implementation of a High-Speed and High-Precision XML Information Retrieval System on Relational Databases, *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML*

Retrieval, INEX 2005, Dagstuhl, Germany, LNCS 3977, Springer-Verlag, (2006), 254-267.

GEVA, S. GPX - gardens point XML information retrieval at INEX 2004. *In:* Fuhr, N., Lalmas, M., Malik, S. and Szlávik, Z. (eds) *Advances in XML Information Retrieval, Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl, Germany*, LNCS 3493, Springer-Verlag, (2005), 211-223.

GOLDMAN, R. & WIDOM, J. DataGuIDes: Enable query formulation and optimization in semistructured databases. *In: Jarke, M. Carey, M.J. Dittich, K.R. Lochovshy, F.H. Loucopoulos, P. Jeusfeld, M.A. (Eds.), Proceedings of the 23rd International Conference on Very Large Data Bases, Rome, Italy,* (1997), 436-445.

GONNET, G., BAEZA-YATES, R. & SNIDER. T. New indices for text: Pat trees and Pat arrays. In: In: Frakes, W. and Baeza-Yates, R (Eds), Information Retrieval: Data Structures and Algorithms, Prentice-Hall, Englewood Cliffs, NJ, (1992), 66–82.

GOU, G., & CHIRKOVA, R., Efficiently Querying Large XML Data Repositories: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19, No. 10, 1381-1403.

HARDING, P.J., LI, Q. & MOON, B. XISS/R: XML Indexing and Storage System Using RDBMS. *In: Freytag, J.C. Lockemann, P.C., Abiteboul, S. Carey, M.J. Selinger, P.G. & Heuer, A. (Eds.), Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany*, (2003), 1073-1076.

HARMAN, D., FOX, E.A., BAEZA-YATES, R. AND LEE, W. Inverted Files. In: Frakes, W. and Baeza-Yates, R (Eds), Information Retrieval: Data Structures and Algorithsm, Prentice-Hall, Englewood Cliffs, NJ, (1992), 28-43.

JONES, S., WALKER, S. GATFORD, M. AND DO. T. Peeling the onion: Okapi system architecture and software design issues, *Journal of Documentation*, 53 (1), (1997), 58-68.

KAUSHIK, R. BOHANNON, P. NAUGHTON, J. & KORTH, H. Covering indexes for branching path queries. *In: Franklin, M.J. Moon, B. & Ailamaki, A, (Eds.), Proceedings of ACM SIGMOD 2002 Conference, Madison Wisconsin, USA,* (2002), 133-144.

LEHTONEN, M. When a few highly relevant answers are enough. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl, Germany*, LNCS 3977, Springer-Verlag, (2006), 296-305.

LEWIS, D., YANG, Y, ROSE, T.G. AND LI, F. RCV1: A new benchmark collection for text categorization research, *Journal of Machine Learning Research*, 5, (2004), 361-397.

LU, W. ROBERTSON, S.E. & MACFARLANE, A. Field-Weighted XML retrieval based on BM25. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl, Germany*, LNCS 3977, Springer-Verlag, (2006), 161-171.

LU, W. ROBERTSON, S.E. & MACFARLANE, A. CISR at INEX 2006. *In: Fuhr, N. Lalmas, M., & Trotman A. (eds), Comparative Evaluation of XML Information Retrieval Systems: Proceedings of the 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006, Dagstuhl, Germany*, LNCS 4518, Springer-Verlag, (2007), 57-63.

MALIK, S., LALMAS, M. FUHR, N. Overview of INEX.2004. *In: Fuhr, N., Lalmas, M., Malik, S. and Szlavik, Z (Eds.), Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl, Germany*, LNCS 3493, Springer-Verlag, (2005), 1-15.

MALIK, S., KAZAI, G., LALMAS, M. AND FUHR, N. Overview of INEX 2005. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl, Germany*, LNCS 3493, Springer-Verlag, (2006), 1-15.

MCHUGH, J. ABITEBOUL, S., GOLDMAN, R., QUASS, D. & WIDOM, J. Lore: A Database Management System for Semistructured Data. *SIGMOD Record, 26(3),* (1997), 54-66.

MILO T, & SUCIU, D. Index structures for path expression. *In: Beeri, C. & Buneman, P. (Eds.), Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel,* (1999), 277-295.

POPOVICI, E., MENIER, G. & MARTEAU, P.F. SIRIUS: A Lightweight XML Indexing and Approximate Search System at INEX 2005. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl, Germany*, LNCS 3977, Springer-Verlag, (2006), 321-335.

ROBERTSON, S.E., LU, W. & MACFARLANE, A. XML-structured documents: retrievable units and inheritance, *In: Legind Larsen, H.; Pasi, G.; Ortiz-Arroyo, D.; Andreasen, T.; Christiansen, H. (Eds.) Proceedings Flexible Query Answering Systems 7th International Conference, FQAS 2006, Milan, Italy, June 7-10, 2006, LNCS, 4027, Springer-Verlag,* (2006), 121-132.

THEOBALD, M., SCHENKEL, R. & WEIKUM, G. TopX and XXL at INEX 2005. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl, Germany*, LNCS 3977, Springer-Verlag, (2006), 282-295.

TROTMAN, A., PHARO, N. & LEHTONEN, M. XML-IR Users and use Cases, *In: Fuhr, N. Lalmas, M., & Trotman A. (eds), Comparative Evaluation of XML Information Retrieval Systems: Proceedings of the 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006, Dagstuhl, Germany*, LNCS 4518, Springer-Verlag, (2007), 400-412.

SAUVAGNAT, K. HLAOUA, L. & BOUGHANEM, M. XFIRM at INEX 2005: Ad-Hoc and Relevance Feedback Tracks. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX*

2005, Dagstuhl, Germany, LNCS 3977, Springer-Verlag, (2006), 88-103.

WANG, H., PARK, S., FAN, W. & YU, P.S. ViST: A Dynamic Index Method for Querying XML Data by Tree Structures. *In: Halevy, A,Y., Ives, Z.G. & Doan, A. (Eds.), Proceedings of ACM SIGMOD 2003 Conference, San Diego, California, USA,* (2003), 110-121.

VAN ZWOL, R. Multimedia strategies for B3-SDR, based on Prinicipal Component Analysis. *In: Fuhr, N. Lalmas, M., Malik, s, Kazai, G. (eds), Advances in XML Information Retrieval: 4th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2005, Dagstuhl, Germany*, LNCS 3977, Springer-Verlag, (2006), 540-553.

WEI, S., & DA-XIN, L, A HybrID method for Efficient Indexing of XML Documents, *In: Lee, S., and Bussler, C. & Shim, S. (Eds), Proceedings of the 2005 International Workshop of Data Engineering Issues in E-Commerce (DEEC05), Los Alamitos, CA,* (2005), 139-143.

WOODLEY, A., GEVA, S. & EDWARDS, S.L., What XML-IR Users May Want, *In: Fuhr, N. Lalmas, M., & Trotman A. (eds), Comparative Evaluation of XML Information Retrieval Systems: Proceedings of the 5th International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2006, Dagstuhl, Germany*, LNCS 4518, Springer-Verlag, (2007), 423-431.

ZOBEL, J. AND MOFFAT, A. Inverted files for text search engines, *ACM Computing Surveys*, Vol. 38, No. 2, (2006), article 6.

Web References

BOSAK, J. Shakespeare's plays in XML. http://www.ibiblio.org/xml/examples/shakespeare. (visited 20th November 2008)

HYREX WEB SITE. http://www.is.informatik.uni-duisburg.de/projects/hyrex/index.html. (visited 20th November 2008)

INEX WEB SITE. http://inex.is.informatik.uni-duisburg.de/. (visited 20th November 2008)

MarkLogic web site. http://www.marklogic.com/. (visited 20th November 2008)

Okapi Documentation. http://soi.city.ac.uk/~andym/OKAPI-PACK/. (visited 20th November 2008)

SOFTWARE AG WEB SITE. Tamino XML database – http://www.softwareag.com/Corporate/products/wm/tamino/default.asp/. (visited 20th November 2008)

TREC CONFERENCE WEB SITE. http://trec.nist.gov. (visited 20th November 2008)