Okapi-based XML indexing

Wei Lu
School of Information Management, Wuhan University, Wuhan City, China, and
Andrew MacFarlane and Fabio Venuti
Department of Information Science, City University London, London, UK

Received 21 November 2008
Revised 27 February 2009
Accepted 14 March 2009

Abstract

Purpose – 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. This paper aims to propose a method for XML indexing based on the information retrieval (IR) system Okapi.

Design/methodology/approach – First, the paper reviews the structure of inverted files and gives 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 the paper explores a revised method implemented on Okapi using path indexing structures. The paper evaluates these index structures through the metrics of indexing run time, path search run time and space costs using the INEX and Reuters RVC1 collections.

Findings – 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.

Practical implications – Overall, the results show the method implemented to support XML search in a traditional IR system such as Okapi is viable.

Originality/value – The paper provides useful information on a method for XML indexing based on the IR system Okapi.

Keywords Information retrieval, Data structures, Extensible markup language, Indexing, Resource efficiency

Paper type Research paper

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 (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.

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.

Aslib Proceedings: New Information Perspectives
Vol. 61 No. 5, 2009
pp. 483-499

0001-253X
DOI 10.1108/00012530910989634

Cooper *et al.* (2001) and 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 and Kossmann (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 (Deutsch *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 and 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 Figure 1 are/newsitem, /newsitem/title, /newsitem/text, /newsitem/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 al.* (2002), Goldman and Widom (1997) and Milo and Suciu (1999) do not support value indexes (Wang *et al.*, 2003). Some systems such as HYREX do now support value centric retrieval (Fuhr and Großjohann, 2004). There has also been a

```
<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>
```

Figure 1.
Prototype of XML record

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 (IR) 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 treat 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. Finally, Section 5 gives a conclusion and outlines further work to be done.

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 I).

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 Figure 1, taken from the Reuters RCV1 collection (Lewis *et al.*, 2004).

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. They cannot, however, represent the complex hierarchical structure of XML documents (such as those in Figure 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 Figure 1, for example, the element "dc" is repeated several times with different attributes: there is no way for a word level index to recognise 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, < metadata > < 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),

Table I. 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

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 two 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, p. 662) assert that "in order to process queries referring to the logical structure of documents" (please refer to Figure 1), "XML query languages must support the following four types of conditions":

(1) Element names: ability to specify element name in search, e.g. from Figure 1, restrict the "dc element" to the value "dc.publisher".

(2) Element index: ability to search on elements, e.g. from Figure 1, the "headline" (field search).

(3) Ancestor/descendant: ability to use the hierarchical structure of the documents for search, e.g. from Figure 1, find the "metadata" then "dc element".

(4) Preceding/following: ability to use the linear sequence of the document for search, e.g. from Figure 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 + tree 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 structure. 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 three main path index files: the path file, the path position file and the path instance offset file. The detailed structure of these files is shown in Figure 2.

Descriptions of each of the files shown in Figure 2 are as follows:

(1) Path file stores the path ID information (see Table II). For each different path, a unique integer ID is given by its occurrence order in the collection (pth_id), together with the path_name and depth. In this file, ix_start and path_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 Figure 1 are/newsitem, /newsitem/title, /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.

(2) Path position file stores path instances' position information in the path instance offset file (see Table III). 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

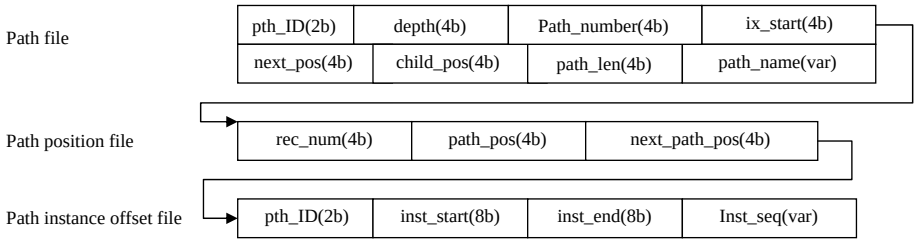

Figure 2.
Structure of main path index files (Nb = N bytes per field, var = variable length field)

Path address	pth_ ID	depth	path_ number	ix_ start	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 II.
Example data for path file

Path position	rec_num	path _ pos	next _ path _ pos
ix_addr1:	1	pi_addr1	0xffffffff
ix_addr2:	1	pi_addr8	0xffffffff
ix_addr3:	1	pi_addr4	0xffffffff
ix_addr4:	1	pi_addr3	0xffffffff
…	…	…	…
ix_addr8:	1	pi_addr11	
		pi_addr12	
		pi_addr13	0xffffffff
ix_addr9:	…	…	…

Table III.
Example data for path position file

specified path are grouped together for improving search speed. This file is similar to a postings file described above.

(3) Path instance offset file stores path instances' position information in the XML collection (see Table IV). In this file, pth_ID is the same as that in path file inst_start and inst_end point to the path instance's start and end positions in the XML collection, and inst_seq is path instance's detailed information which contains element index information. For example, given a 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 inst_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 Figure 1, then see the following Tables II-IV for the data in these files (in the example *addr* is the record start position in the path index files).

Taking path "/newsitem" as an example, it occurs at the beginning of the collection, so the path_ID is set to 1. The 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 0xffffffff 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 inst_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,

Offset address	pth_ID	inst_start	inst_end	inst_seq
pi_addr1:	1	0	456	1
pi_addr2:	2	...	...	11
pi_addr3:	3	...	...	11
pi_addr4:	4	...	...	11
...	...			
pi_addr11:	11	...	...	111
pi_addr12:	11	...	...	112
pi_addr13:	11	...	...	113
...	...			

Table IV. Example data for path instance offset file

are proposed to create an index on the elements for all paths in the path file. The structure of these two files is shown in Figure 3.

Descriptions of each of the files shown in Figure 3 are as follows:

- 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 Tables II and III, respectively).

Thus, for a vague join path search such as "newsitem//dc", we could easily split this path into two elements "newsitem" and "dc". For each element, we could obtain a result path set where the elements occur by retrieving data from both file files shown in Figure 3. Further, the integer value elem_depth could be used to join 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 comparison by using elem_depth, which could improve the search speed. The path centric search evaluation is provided in Section 4. Figure 4 shows the 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. This structure, however, 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 I 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

Element file:

[<elem_id(2)><elem_start(4)><elem_num(2)><elem_len(2)><elem_name(var)>]

Element_position file:

[<path_id(2b)><elem_depth(2)>]

Figure 3.
Further path index files

```
Procedure: Path_indexing(D, R, P, E,B)
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 themto Element position file
      Group and sort all temp Path position file and merge to final Path position file
End
```

Figure 4.
Path indexing algorithm

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. Figure 5 shows both the path and value search process. Details of how these structures support ranking through the BM25 F function can be found in Lu *et al.* (2006, 2007).

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

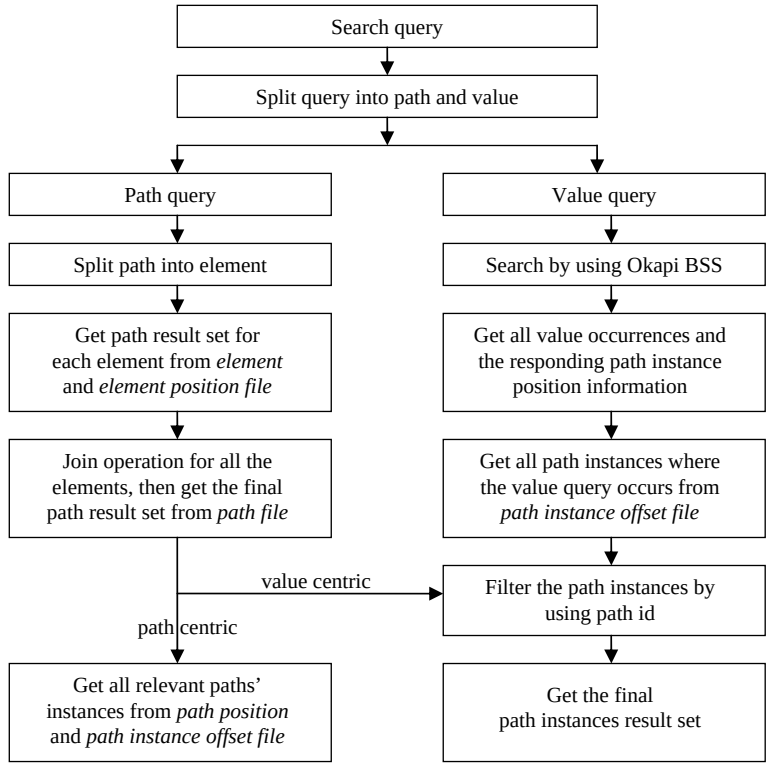

Figure 5.
Both path and value search process

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):

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

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

(3) Shakespeare's Plays: this data set contains the 37 plays of Shakespeare marked up in XML format.

(4) Reuters RCV1 collection: a set of newswire articles from Reuters split into subsets to test scalability.

Tables V and VI give more details on the various statistics on these collections.

4.2 Experimental results

Figure 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 494MB, 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

Data sets	INEX 1.4	INEX 1.6	Shakespeare works
Size of data (MB)	494	705	9.99
No. of elements	8,239,873	11,411,135	21
No. of attributes	2,204,688	4,669,699	179,689
No. of records	12,107	16,819	37
Avg path level	8	8	5

Table V.
Benchmark parameters (INEX and Shakespeare collections)

Data sets	Reuters 1	Reuters 2	Reuters 3	Reuters 4	Reuters 5	Reuters 6
Size of data (MB)	250	500	750	1000	1500	2000
No. of elements	3,491,446	6,937,705	1,0362,907	13,802,112	20,703,163	26,998,953
No. of attributes	3,659,853	7,334,929	10,954,155	14,590,239	21,832,916	28,445,398
No. of records	89,114	178,121	267,052	355,060	531,744	692,874
Avg path level	6	6	6	6	6	6

Table VI.
Benchmark parameters (Reuters RCV1 subsets)

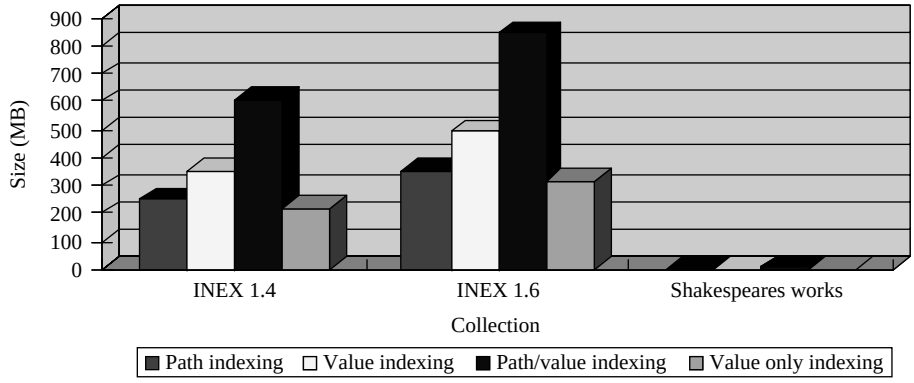

Figure 6.
Comparison of index size

which nearly doubles the size of the value index size and we create a path position file for locating each path instance in the path instance offset file.

Figure 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 per cent than that of the value only indexing method. For small collections such as the Shakespeare Works, the indexing completes very quickly.

To investigate the scalability of indexing with the growth of the data collection's 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 Figures 8 and 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. The revised total index size is three times larger than that of the original inverted file 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 Figures 8 and 9 shows 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 VII 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

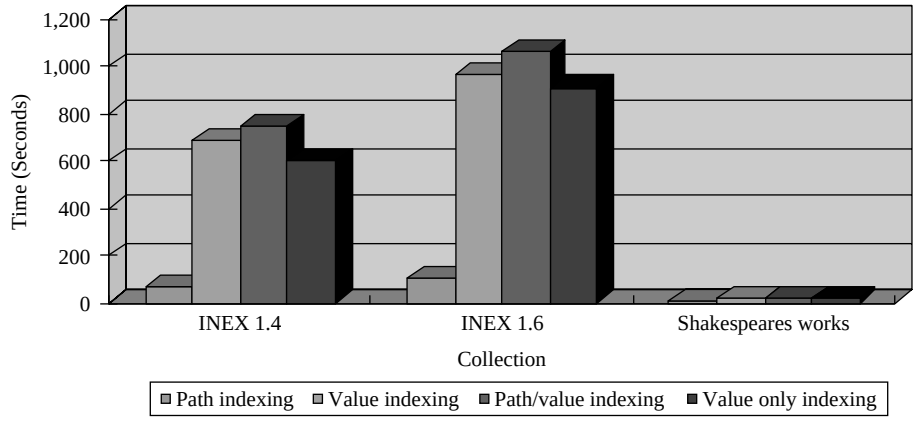

Figure 7.
Comparison of indexing run times

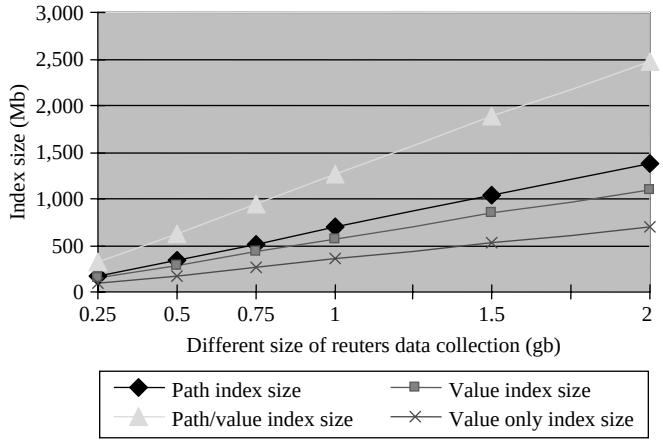

Figure 8.
Scalability of the indexing using subsets of the Reuters RCV1 collection (size)

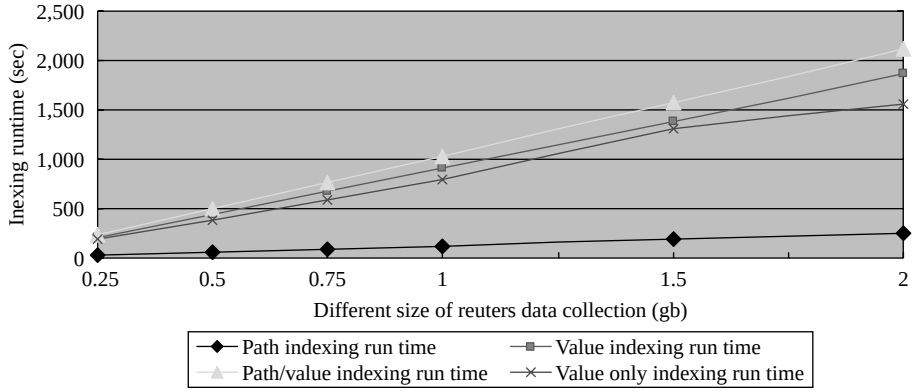

Figure 9.
Scalability of the indexing using subsets of the Reuters RCV collection (run-time)

Path type	Path query	No. of relevant path	No. of relevant path instance	Cost time display (millisecond)	Cost time no display (millisecond)
Absolute path	/article/bdy/sec	1	65407	7	0.09
	/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 VII.
Path search experiment on INEX 1.4 data set

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 related to the occurrences of the element in the collection. For example, both "bdy" and "sec" occur so often in the

collection that any join operation is therefore more expensive. The average search time on 50 randomly selected queries for absolute path and vague path, respectively, are shown in Table VIII. Results show that the path only search is quite efficient, and also the path index structure is quite flexible in supporting any kind of query.

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.

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 content only (CO) 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, for example, 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.

Table VIII.
Fifty 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

References

Bosak, J. (2006), "Shakespeare's Plays in XML", available at: www.ibiblio.org/xml/examples/shakespeare (accessed 22 January 2009).

Chung, C., Min, J. and Shim, K. (2002), "APEX: an adaptive path index for XML data", in Franklin, M.J., Moon, B. and Ailamaki, A. (Eds), *Proceedings of ACM SIGMOD 2002 Conference, Madison Wisconsin, USA*, ACM Press, New York, NY, pp. 109-20.

Cooper, B.F., Sample, N., Franklin, M.J., Hjaltason, G.R. and Shadmon, M. (2001), "A fast index for semistructured data", in Apers, M.G., Atzeni, P., Ceri, S., Paraboschi, S., Ramamohanaroa, K. and Snodgras, R.T. (Eds), *Proceedings of the 27th International Conference on Very Large Data Bases, Rome, Italy*, Morgan Kaufmann, San Francisco, CA, pp. 341-50.

Deutsch, A., Fernandez, M. and Suciu, D. (1999), "Storing semistructured data with STORED", in Delis, A., Faloutsos, C. and Chandeharizadeh, S. (Eds), *Proceedings of ACM SIGMOD Conference 1999, Philadelphia, Pennsylvania, USA*, ACM Press, New York, NY, pp. 431-42.

Florescu, D. and Kossmann, D. (1999), "Storing and querying XML data using an RDMBS", *IEEE Data Engineering Bulletin*, Vol. 22 No. 3, pp. 27-34.

Fuhr, N. and Govert, N. (2002), "Index compression vs. retrieval time of inverted files for XML documents", *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA*, ACM Press, New York, NY, pp. 662-4.

Fuhr, N. and Großjohann, K. (2004), "XIRQL: an XML query language based on information retrieval concepts", *ACM Transactions on Information Systems*, Vol. 22 No. 2, pp. 313-56.

Fujimoto, K., Shimizu, T., Terada, N., Hatano, K., Suzuki, Y., Amagasa, T., Kinutani, H. and Yoshikawa, M. (2006), "Implementation of a high-speed and high-precision XML information retrieval system on relational databases", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 254-67.

Geva, S. (2005), "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, Berlin, pp. 211-23.

Goldman, R. and Widom, J. (1997), "DataGuides: enable query formulation and optimization in semistructured databases", in Jarke, M., Carey, M.J., Dittich, K.R., Lochovshy, F.H., Loucopoulos, P. and Jeusfeld, M.A. (Eds), *Proceedings of the 23rd International Conference on Very Large Data Bases, Rome, Italy*, Morgan Kaufmann, San Francisco, CA, pp. 436-45.

Gonnet, G., Baeza-Yates, R. and Snider, T. (1992), "New indices for text: Pat trees and Pat arrays", in Frakes, W. and Baeza-Yates, R. (Eds), *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, pp. 66-82.

Gou, G. and Chirkova, R. (2007), "Efficiently querying large XML data repositories: a survey", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 19 No. 10, pp. 1381-403.

Harding, P.J., Li, Q. and Moon, B. (2003), "XISS/R: XML indexing and storage system using RDBMS", in Freytag, J.C., Lockemann, P.C., Abiteboul, S., Carey, M.J., Selinger, P.G. and Heuer, A. (Eds), *Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany*, Morgan Kaufmann, San Francisco, CA, pp. 1073-6.

Harman, D., Fox, E.A., Baeza-Yates, R. and Lee, W. (1992), "Inverted Files", in Frakes, W. and Baeza-Yates, R. (Eds), *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, pp. 28-43.

Jones, S., Walker, S., Gatford, M. and Do, T. (1997), "Peeling the onion: Okapi system architecture and software design issues", *Journal of Documentation*, Vol. 53 No. 1, pp. 58-68.

Kaushik, R., Bohannon, P., Naughton, J. and Korth, H. (2002), "Covering indexes for branching path queries", in Franklin, M.J., Moon, B. and Ailamaki, A. (Eds), *Proceedings of ACM SIGMOD 2002 Conference, Madison Wisconsin, USA*, ACM Press, Madison, WI, pp. 133-44.

Lehtonen, M. (2006), "When a few highly relevant answers are enough", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 296-305.

Lewis, D., Yang, Y., Rose, T.G. and Li, F. (2004), "RCV1: a new benchmark collection for text categorization research", *Journal of Machine Learning Research*, Vol. 5, pp. 361-97.

Lu, W., Robertson, S.E. and MacFarlane, A. (2006), "Field-weighted XML retrieval based on BM25", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 161-71.

Lu, W., Robertson, S.E. and MacFarlane, A. (2007), "CISR at INEX 2006", in Fuhr, N., Lalmas, M. and 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, Berlin, pp. 57-63.

McHugh, J., Abiteboul, S., Goldman, R., Quass, D. and Widom, J. (1997), "Lore: a database management system for semistructured data", *SIGMOD Record*, Vol. 26 No. 3, pp. 54-66.

Malik, S., Lalmas, M. and Fuhr, N. (2005), "Overview of INEX.2004", in Fuhr, N., Lalmas, M., Malik, S. and Szlavik, Z. (Eds), *Advances in XML Information Retrieval: 3rd International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Dagstuhl, Germany*, LNCS 3493, Springer-Verlag, Berlin, pp. 1-15.

Malik, S., Kazai, G., Lalmas, M. and Fuhr, N. (2006), "Overview of INEX 2005", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 1-15.

Milo, T. and Suciu, D. (1999), "Index structures for path expression", in Beeri, C. and Buneman, P. (Eds), *Proceedings of the 7th International Conference on Database Theory, Jerusalem, Israel*, LNCS 1540, Springer-Verlag, Berlin, pp. 277-95.

Popovici, E., Menier, G. and Marteau, P.F. (2006), "SIRIUS: a lightweight XML indexing and approximate search system at INEX 2005", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 321-35.

Robertson, S.E., Lu, W. and MacFarlane, A. (2006), "XML-structured documents: retrievable units and inheritance", in Legind Larsen, H., Pasi, G., Ortiz-Arroyo, D., Andreasen, T. and Christiansen, H. (Eds), *Proceedings Flexible Query Answering Systems 7th International Conference, FQAS 2006, Milan, Italy, June 7-10, 2006*, LNCS, 4027, Springer-Verlag, Berlin, pp. 121-32.

Sauvagnat, K., Hlaoua, L. and Boughanem, M. (2006), "XFIRM at INEX 2005: ad-Hoc and relevance feedback tracks", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 88-103.

Theobald, M., Schenkel, R. and Weikum, G. (2006), "TopX and XXL at INEX 2005", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 282-95.

Trotman, A., Pharo, N. and Lehtonen, M. (2007), "XML-IR users and use cases", in Fuhr, N., Lalmas, M. and 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, Berlin, pp. 400-12.

van Zwol, R. (2006), "Multimedia strategies for B3-SDR, based on principal component analysis", in Fuhr, N., Lalmas, M., Malik, S. and 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, Berlin, pp. 540-53.

Wang, H., Park, S., Fan, W. and Yu, P.S. (2003), "ViST: a dynamic index method for querying XML data by tree structures", in Halevy, A.Y., Ives, Z.G. and Doan, A. (Eds), *Proceedings of ACM SIGMOD 2003 Conference, San Diego, CA, USA*, ACM Press, New York, pp. 110-21.

Wei, S. and Da-Xin, L. (2005), "A hybrid method for efficient indexing of XML documents", in Lee, S., Bussler, C. and Shim, S. (Eds), *Proceedings of the 2005 International Workshop of Data Engineering Issues in E-Commerce (DEEC05), Los Alamitos, CA*, IEEE, Los Alamitos, CA, pp. 139-43.

Woodley, A., Geva, S. and Edwards, S.L. (2007), "What XML-IR users may want", in Fuhr, N., Lalmas, M. and 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, Berlin, pp. 423-31.

Zobel, J. and Moffat, A. (2006), "Inverted files for text search engines", *ACM Computing Surveys*, Vol. 38 No. 2, p. 6.

Further reading

HYREX website (2009), www.is.informatik.uni-duisburg.de/projects/hyrex/index.html (accessed 22 January 2009).

INEX web site (2009), http://inex.is.informatik.uni-duisburg.de/ (accessed 22 January 2009).

MarkLogic website (2009), www.marklogic.com/ (accessed 22 January 2009).

Okapi Documentation (2009), http://soi.city.ac.uk/~andym/OKAPI-PACK/ (accessed 22 January 2009).

Software AG website (2009), Tamino XML database. www.softwareag.com/Corporate/products/wm/tamino/default.asp/ (accessed 22 January 2009).

TREC Conference website (2009), http://trec.nist.gov (accessed 22 January 2009).

Corresponding author
Andrew MacFarlane can be contacted at: andym@soi.city.ac.uk