Structured Information Retrieval in XML documents

Evangelos Kotsakis
Joint Research Center (CCR), TP261,
I-21020 Ispra (VA), Italy
kotsakis@acm.org

ABSTRACT

Query languages that take advantage of the XML document structure already exist. However, the systems that have been developed to query XML data explore the XML sources from a database perspective. This paper examines an XML collection from the viewpoint of Information Retrieval (IR). As such, we view the XML documents as a collection of text documents with additional tags and we attempt to adapt existing IR techniques to achieve more sophisticated search on XML documents. We employ a class of queries that support path expressions and suggest an efficient index, which extends the inverted file structure to search XML documents. This is accomplished by integrating the XML structure in the inverted file by combining the inverted file with a path index. The proposed structure is a lexicographical index, which may be used for the evaluation of queries that involve path expressions. Moreover, this paper discusses a ranking scheme based on both the term distribution and document structure. Some performance remarks are also presented.

Keywords

XML information retrieval, Web data indexing, semistructured data indexing, full text searching

1. INTRODUCTION

The *eXtensible Markup Language* (XML)[2] is a universal format for data exchange on the Web and in the near future we will find large XML document collections on the Web. As a result, it has become crucial to address the question of how we can efficiently query and search large corpora of XML documents.

To date, most research on storing, indexing and querying XML documents has been based on the work on semistructured data [13]. Models for representing XML data have been proposed from a database perspective and they have been tailored to facilitate querying processing on semistructured data [8]. XML databases, which are based on such models, merge diverse XML elements from distinct XML sources into a single hierarchy and then use this internal representation to query XML data. Such approaches overlook the notion of text document and view an XML corpus as a collection of distinct XML elements. While this approach is sufficient for a variety of applications that require assembling different semi-structured sources into a single database, it may cause difficulties in answering Boolean and ranked queries. Such queries have been extensively used in information retrieval systems and they consist of a list of terms or sample of text. A simplified form of such queries is the following: *"find all documents in the collection that best match a given description"*. In the context of XML documents, Boolean queries may be also enriched by path expressions so that the "description" clause may be a conjunctive or disjunctive sequence of terms or path expressions. If such queries are to be evaluated on an XML corpus, an alternative approach is required that views the corpus as a collection of text documents with additional tags. Such an approach might also support the necessary indexing mechanism that facilitates fast response.

Existing XML data processing systems explore the XML sources from the database viewpoint. The result set of such queries consists of XML elements that precisely satisfy the query conditions. In an XML information retrieval system, answering such a query is not sufficient for finding relevant documents. Extra support is needed in both ranking and indexing to achieve this goal.

The proposed index structure combines an inverted file index with a path index in order to facilitate the evaluation of Boolean queries. This enables more sophisticated search on the structure as well as the content of the XML documents, while similarity and ranking may be employed to leverage keyword search. In particular, ranking is relied on the structure of the XML documents.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 discusses the index organization. Query evaluation is discussed in section 4. Section 5 presents the ranking scheme and section 6 discusses some performance aspects of the system. Section 7 summarizes the contributions and concludes the paper by providing some directions for future work.

2. RELATED WORK

In this section we briefly review previous approaches to the problem of indexing structured and semistructured data. Many approaches, which have been mainly proposed by the database community, are close to the research done on semistructured databases [8, 13, 14, 4]. In this case, the main

SAC 2002, Madrid, Spain

objective is to build semistructured management systems that facilitate query processing. Several query languages have been proposed for this purpose. For an overview, the reader may refer to [6, 1]. Other approaches are stemmed by the work done on structured documents from the perspective of information retrieval [12, 19, 11, 16, 17].

Among the database approaches is the Lore system [13]. Lore accomplishes the uploading of new documents by adding the elements of the documents in a tree like structure and updating several indexes (value index, text index, link index and path index [14]). From that point on, any data access is performed by considering the whole database as a huge tree that contains XML elements. Lore approach seems to view an XML document as a database and a set of documents as a single large database where all documents are mixed together into a tree like structure.

In our approach a set of XML documents is handled as is (set of documents) and the index facilitates the retrieval of XML documents that match better the query terms. The output of the query evaluation is a set of XML documents. The proposed approach is closer to the Information retrieval end and the objective is to retrieve the XML documents that match the user preferences expressed by way of Boolean query terms.

Indexing structures for documents are discussed in [7] and for structured documents in [12]. Inverted files and signature files have been used only for searching literal terms. If queries contain structured elements (paths) an additional index that captures the structure of the documents is needed. In [12], complete trees are used for this purpose. While this approach works for exact matching, it does not for partial matching.

A query evaluation scheme called BUS and an indexing structure for retrieving structured documents has been proposed in [19]. In this approach indexing is performed at the leaf element and the query evaluation computes the similarity by accumulating the weights in a bottom up way. Index organized tables are proposed in [11], which actually implements posting information by using the BUS strategy.

3. INDEX ORGANIZATION

The first processing step in organizing the index structure is the term recognition shown in figure 1. On this step, we need to specify what terms must be indexed in the XML document. This is accomplished by using some normalization algorithm that removes unimportant terms (stop words) [7, 15]. This may involve checking each word against a stop-list of common words. If the word is not a common one, it may be additionally passed through a stemming algorithm. The resultant stems are recorded together with the associated path. The last processing phase of this step involves the estimation of the distribution ("*within document frequency*") of all terms (literals and tags). Any path and term duplication is removed so that the resulting XML summary tree contains each path at most once. In principle, the summary tree is smaller than the original XML document and it contains only those literal terms and tags that are important as far as indexing is concerned. Moreover, the summary tree keeps the structure of the original XML document facilitating in that way advanced search within the content of the tags. The term distribution may be in addition used to generate weights for each term in the summary tree in order to facilitate ranking. Summary trees constitute the input to

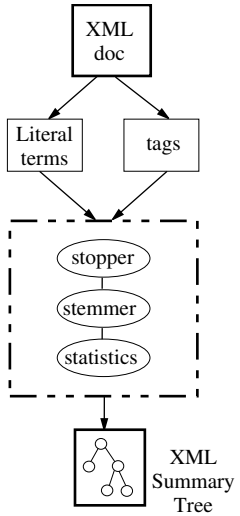

Figure 1: XML document normalization process

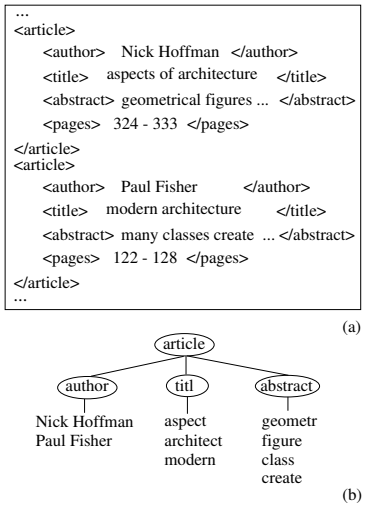

Figure 2: (a) An XML document and (b) its Summary tree

the index structure.

Figure 2(a)shows an XML document and figure 2(b)shows the resulting XML summary tree. Note, that each path appears once in the summary tree and each literal term and tag has been replaced by its stem. Terms or tags that have no indexing value (i.e. the <pages> tag) are not present in the summary tree.

The second processing step deals with the loading of the summary trees into the index structure. This involves the separation of content data from path data (figure 3). Content data is raw text aimed to be stored in the inverted file and path data is structured text aimed to be stored in the path index. The path index is a hierarchy of tags, which records every single path in the collection. A list of documents (posting list) is also stored alongside each entry of the inverted file. Each entry of the posting list usually contains the document id and a reference to a node (tag) of the path index. This node is the last node in the path that con-

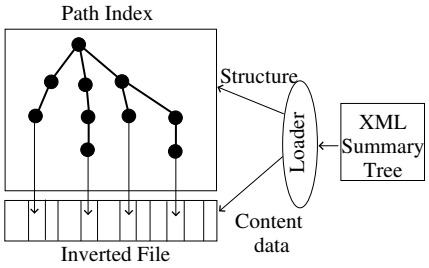

Figure 3: Loading an XML summary tree into the index structure

tains the term. Other data may be also stored in a posting list entry for facilitating ranking. Such data may include a manually or automatically assigned term weight, which may be used to estimate the document relevance to a given path query.

Deletions and Insertions of summary trees may reduce or expand the index structure respectively, reflecting in this way the removal and insertion of XML documents. When a document is added to the collection, its associated summary tree is inserted to the index structure. When a document is removed form the collection, the corresponding summary tree is also removed from the index structure. A document update may be realized through a sequence of remove and insert operations. Initially (before any document insertion) the index structure is empty. Algorithm 1 inserts a new summary tree in the index structure by storing the structured part of the summary tree into the path index and the literal part of the element content into the inverted file.

ALGORITHM 1. Insert (T, P, I)

Input: *T is a pointer to the root node of the summary tree to be inserted in the index structure. The index consists of a list of literal terms (inverted file) referenced by I, and a path index whose root is referenced by P.*

Output: *The index structure containing the new summary tree.*

Method: *Inserts the new summary tree T*

I1 *If the index structure is empty (no summary tree has ever been inserted) a new root is created, referenced by P. The function tag(N) returns the tag of the node pointed to by N. Then the recursive function in step I2 is invoked by AddSummaryTree(T,P).*

I2 $AddSummaryTree(t, p)$
/* *t and p are pointers to nodes in trees T and P respectively* */
if *there is no child c of p such that* $tag(c) = tag(t)$
then {
make a new child node c of p such that $tag(c) = tag(t)$
$UpdateInvertedFile(I, t, c)$ }
for *each child x of t* **do** $AddSummaryTree(x, c)$

$UpdateInvertedFile(I, t, c)$ invokes a procedure that stores the literal content (not sub-elements) of the node t into the inverted file I by adding all the terms to the vocabulary of the inverted file and updating all the necessary entries in the inverted lists. Moreover, this procedure makes a link between the node c of the path tree, for which tag(c)=tag(t), and each newly inserted literal term of the inverted file. This particular reference associates a term with the path in the document where the term is located. The inverted file update is not discussed here. This is fully presented in [10], which also includes a few modifications that facilitate the creation of very large inverted files . The $UpdateInvertedFile$ procedure is used as an interface for updating the inverted file and creating the necessary links to the path index. This is as follows:

UpdateInvertedFile(I, t, c)

Comments: $content(t)$ is the literal content of the tag t in the summary tree. I is the inverted file where the terms of $content(t)$ will be stored. c is the node in the path index to which all the terms in $content(t)$ will be linked.

Method: Updates the inverted file.
For each term x in the $content(t)$ **do** {
If x is not in the vocabulary of I **then**
add x to I.
Update accordingly the inverted list of the term x.
Make a reference from c to the entry x}

The deletion of a document from the index is performed by deleting the entries from the inverted lists. If the document entry is the last in the inverted list, the term is also removed from the vocabulary. If after removing the document entries there are nodes in the path index without any link to an inverted list entry, these nodes are also removed from the path index.

ALGORITHM 2. Delete(docID)

Input: *The docID of the document that will be removed from the index.*

Output: *index after removing the document*

Method: *Deletes the document from the index.*
For *each term x in the vocabulary of the inverted file that can be identified as being a term of the document with ID docID* **do**{
Delete the contribution of x in the inverted file.
Update the path index to the root by removing any link to x. }

4. QUERY EVALUATION

A search in the index consists of two steps:

1. A search for identifying the XML documents containing the path of the query term.

2. A search for identifying the XML documents containing the raw text of the query term.

Upon receiving a query, the query evaluation process decomposes the query into its constituent conjunctive and disjunctive terms. Each query term consists of a path and raw text (literal part). The path is checked against the path index of the index structure. If there is a match, all the inverted lists

are identified by generating a candidate list A of documents and a candidate list T of vocabulary terms that happens to be at the end of this path. The literal part is then checked against the candidate terms in T and for those terms that there is a match the document entries are retrieved from the inverted lists and a second candidate set of documents B is generated. A ranking algorithm is also applied at this point to rank the documents in B based on frequency information held in the entries of the inverted lists. The answer is the intersection of A and B.

The following Algorithm 3 shows how to evaluate a conjunctive Boolean path query.

ALGORITHM 3. Query evaluation (Q)

Input: *The query Q with $x_1, x_2, \ldots, x_n$ conjunctive terms.*

Output: *A set S of documents.*

Method: *Normalize the query terms $x_1, \ldots, x_n$ and set $S = \emptyset$. Decompose the query into several conjunctive terms* **For** *each term $x_i, (1 \leq i \leq n)$ of the query* **do** { *Let T_i be the set containing the inverted lists that match the path of the term x_i. Let A_i be the set containing the document entries that match the path of the term x_i. Extract from T_i the list of documents pointed to by the literal part of x_i and store this list to the set B_i.}*
The result set S is given by

$$S = \bigcap_{i=1}^{n} (A_i \cap B_i) \tag{1}$$

In the case of a disjunctive query, S is given by

$$S = \bigcup_{i=1}^{n} (A_i \cap B_i) \tag{2}$$

5. RANKING

This section describes the ranking approach to term weighting. The proposed approach is simple and involves no complexity in implementation. Ranking does not only rely on the term weight based on the term distribution both within the XML document and the entire XML collection, but also on the structural position of the term.

As such, the ranking scheme is divided into two components. The first one defines the term weight in terms of its distribution and the second one in terms of its structural position. The final weight of a term is composed by multiplying the two estimates. The first component evaluates the weight as if the term was from an unstructured document. It has been shown that the most important measure is the *Inverse Document Frequency* (IDF) [20, 9]. We have adopted the IDF definition in [5], which for a given term i is as follows:

$$IDF_i = log_2 \frac{N - n_i}{n_i} \tag{3}$$

Where N is the number of XML documents in the collection and n_i is the number of occurrences of the term i in the collection. Based on the above measure, the weight of the term i in the XML document j is given by

$$w_{ij} = freq_{ij} \times IDF_i \tag{4}$$

where $freq_{ij}$ is the *within document frequency* of term i in the document j.We refer to the measure w_{ij} as *statistical weight*.

The second component evaluates the term weight according to its position in the XML document by introducing a coefficient for each single path in the collection. This is accomplished by assigning such a coefficient to each node in the path index tree. The coefficient specifies the importance of the path context from the root to the tag where a term might appear. For example, any literal term at the end of the path 'journal/issue/article/title' may have larger coefficient than a term at the end of the path 'journal/issue/article/abstract'. This is to show that a term that appears in titles is more important than a term in abstracts.

Let f_p be the coefficient assigned to the path p at the end of which the term i is found and w_{ij} is the statistical weight of the term i in the document j . The final weight of a term i in document j is given by $f_p * w_{ij}$.

6. PERFORMANCE

As experimental data, we used the Cystic Fibrosis (CF) document collection, which represents a subset of MEDLINE data [18]. The original CF collection has been transformed into an XML document collection consisting of 1239 XML documents, which share the same Document Type Definition (DTD). The size of the CF XML collection is 6MB. This collection comes with 100 queries bundled with their correct answers. All the tests were accomplished on a Pentium III personal computer with 256MB RAM. The index overhead in size is 2.5MB and the index can be built in less than a minute. During retrieval, the accuracy of the responses to the given queries is very high. The ranking given by the proposed system matches the score of the given results for every query. The response time for each query depends on the number of terms and on the length of the posting list of the term. However for the given queries, the response time is between 0.10 and 0.25 sec. The average response time of the 100 queries is 0.135 sec.

7. CONCLUSIONS AND FUTURE WORK

The indexing process generates an inverted file index and a path index containing all possible paths in the collection. The novelty and advantages of the proposed index are summarized as follows:

1. It combines seamlessly two indexes; an inverted file and a path index
2. The path index contains normalized tags. This feature may facilitate similarity search by content and structure.
3. Both the path index and inverted file are expanded and reduced dynamically by adding and removing XML documents.

The proposed indexing structure is powerful, easy to implement and maintain. The use of an index structure that combines an inverted index and a path index improves search efficiency for large collections of XML documents.

The presence of markup tags in XML documents suggests that indexing the tags, or a normalized form of it, might be an effective approach to capturing document structure. However, there are several problems with this approach. Different XML documents may use different tags to describe the same piece of information. It is really difficult, if not

impossible, to form a standard description for every single piece of data. XML has been actually introduced to describe irregular data and as such the problem of having different tags to describe the same piece of information is quite inherent to the XML language. It is then desirable to provide lists of equivalent tags. Tags with a high degree of semantic proximity must be handled as if they were the same. Synonymous tags (like "journal" and "periodical") must be indexed as if they were the same. Tag transformation might be useful in order to handle variant words using sound-like methods. Eliminating suffixes and prefixes from the tags may further facilitate the content regularity. Devising techniques for handling proximity measures between tags can further increase the accuracy of XML information retrieval. Further research is planed towards tag similarity searching in order to address the problem of proximity searching and achieve more effective XML document ranking.

8. REFERENCES

[1] Angela Bonifati, Stefano Ceri. Comparative Analysis of Five XML Query Languages, ACM SIGMOD Record, **29**(1): 68-79 (2000).

[2] Tim Bray, Jean Paoli and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0, *W3C Recommendation*, available at http://www.w3.org/TR/1998/REC-xml-19980210.

[3] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *proceedings of the ACM-SIGIR International Conference on Research and Development in Information Retrieval*, Montreal, Canada, pp. 97-110, (June 1985).

[4] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi. XML: Current Developments and Future Challenges for the Database Community. In *Proc. of the 7th International Conference on Extending Database Technology (EDBT 2000)*, pp 3-17, Konstanz, Germany (2000).

[5] W. B. Croft and D. J. Harper. Using Probabilistic Models of Document Retrieval without Relevance Information. *Journal of Documentation*, **35**(4):285-295, (1979).

[6] Alin Deutsch, Mary F. Fernandez, Daniela Florescu, Alon Y. Levy, David Maier, Dan Suciu. Querying XML Data. *IEEE Data Engineering Bulletin*, **22**(3):10-18 (1999).

[7] W. Frakes and R. Baeza-Yates (eds). *Information Retrieval: Algorithms and Data Structures*. Prentice-Hall (1992).

[8] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimazation in Semistructured Databases, In *Proceedings of the 23rd VLDB Conference*, pp. 436-445, Athens, Greece, August 25-29, (1997).

[9] D. K. Harman. Ranking Algorithms. In *Information Retrieval: Data Structures and Algorithms*, W. B Frakes and R. Baeza-Yates (Eds) Prentice-Hall, Englewood Cliffs, N.J. pp. 363-392 (1992)

[10] D. K. Harman, E. A. Fox, R. Baeza-Yates and W. C. Lee. Inverted files. In *Information Retrieval: Data Structures and Algorithms*, W. B Frakes and R. Baeza-Yates (Eds). Prentice-Hall, Englewood Cliffs, N.J. pp. 28-43 (1992)

[11] Hyunchi Jang, Youngil Kim and Dongwook Shin. An effective mechanism for index update in structured documents. In *Proceedings of the eighth international conference on Information knowledge management(CIKM'99)*, pp. 383 - 390 (1999).

[12] Yong Kyu Lee, Seong-Joon Yoo, Kyoungro Yoon and P. Bruce Berra. Index structures for structured documents. In *Proceedings of the 1st ACM international conference on Digital libraries (DL'96)*, pp. 91 - 99 (1996)

[13] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, Jennifer Widom. Lore: A Database Management System for Semistructured Data. *ACM SIGMOD Record* **26**(3): 54-66 (1997).

[14] J. McHugh, J. Widom, S. Abiteboul, Q. Luo, and A. Rajaraman. Indexing Semistructured Data. *Technical Report*, Computer Science Dept., Stanford University (1998)

[15] Andrei Mikheev. Document Centered Approach to Text Normalization. In *Proceedings of the Annual ACM Conference on Research and Development in Information Retrieval (SIGIR'00)*, pp. 136-143 Athens, Greece (2000).

[16] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.* **14**(4):349 - 379 (Oct. 1996).

[17] Gonzalo Navarro and Ricardo Baeza-Yates. Proximal nodes: a model to query document databases by content and structure. *ACM Transactions on Information Systems*, **15**(4): 400 - 435 (Oct. 1997)

[18] W. M. Shaw, J.B. Wood, R.E. Wood and H.R. Tibbo. The Cystic Fibrosis Database: Content and Research Opportunities. *Library and Information Science Research (LISR)*, **13**: 347-366 (1991).

[19] Dongwook Shin, Hyuncheol Jang, Honglan J. BUS: An Effective Indexing and Retrieval Scheme in Structured Documents . In *proceedings of the third ACM Conference on Digital libraries (DL'98)* pp. 235-243 (1998).

[20] Jones K. Sparck. A Statistical Interpretation of Term Specificity and its Application in Retrieval. *Journal of Documentation* **28**:11-21, (1972).

About the author

Evangelos Kotsakis received his B.Sc. degree in computer science from the University of Athens, Greece in 1993, his M.Sc. and Ph.D degrees in engineering from the University of Salford, England in 1994 and 1998 respectively. From 1998 to 1999, he worked on space applications in the Joint Research Center, Ispra, Italy. From 1999 to 2000, he held visiting posts in the Federal Institute of Technology (ETH), Zurich, Switzerland and VTT Information Technology, Helsinki, Finland. He is currently a research associate in the Joint Research Center at Ispra, Italy. His research interests include Web data management, XML data processing, semistructured databases, data warehousing, data mining and mobile data management.