On Efficient Part-match Querying of XML Data*

Michal Krátký, Marek Andrt

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba
{michal.kratky,marek.andrt}@vsb.cz
Czech Republic

Abstract. The XML language have been becoming de-facto a standard for representation of heterogeneous data in the Internet. From database point of view, XML is a new approach to data modelling. Implementation of a system enabling us to store and query XML documents efficiently (so called native XML databases) require a development of new techniques. The most of XML query languages are based on the language XPath and use a form of path expressions for composing more general queries. These languages make it possible a part-match querying string values of elements and attributes. Particularly, such queries are common for document-centric XML documents. The document-centric documents are often widely unstructured, contain the mixed content and so on. In particular, such documents are (after a transformation to well-formed XML documents) entire information of broad Web. Previously published multi-dimensional approaches to indexing XML data use paged and balanced multi-dimensional data structures. In the paper we extend the approach for the part-match querying XML data.

Key words: XML, indexing XML data, multi-dimensional data structures, part-match querying, XPath, XQuery

1 Introduction

The mark-up language *XML (eXtensible Markup Language)* [22] is recently understood as a language for data representation. Important properties of the language are heterogeneity, extensibility, and flexibility. From database point of view, the XML is a new approach to data modelling [18]. A *well-formed* XML document or a set of documents is an XML database and the associated DTD or schema specified in the language *XML Schema* [23] is its database schema. Implementation of a system enabling us to store and query XML documents efficiently (so called *native XML databases*) requires a development of new techniques [18]. A number of languages have been developed for querying over XML data e.g., *XML-QL* [8], *XPath* [21], and *XQuery* [20]. The common feature of such languages is the usage of regular path expressions for formulation of the

* Work is partially supported by Grant of GACR No. 201/03/0912.

path in the graph modelling an XML document. Such a path is a sequence of element or attribute names from the root element to a leaf.

The XML data are instance of *semistructured* data. An unstructured data may occur into the structured elements. Further, an XML document can be classified on the basis of contained data, as a *data-centric* or *document-centric*. The data-centric XML documents have got well defined regular structure and capture a structured data, e.g. forms. Such data is often possible to map in a set of relations [17]. On the other hand, the *document-centric* documents are often much unstructured, contain fewer elements with amount of unstructured data (e.g. an XML database of articles). However the most of XML documents are combined from both types (so-called *hybrid* documents).

Languages as the XQuery and XPath contain many constructs for querying both data-centric and document-centric XML documents. Such languages provide structures for part-match querying values of elements and attributes. For example, the XPath language allows a filtration unmeant elements in a result set using the *predicate filter*. One from the predicate filters is filter category applicable to string values of elements. For example, the function *contains()* provides a selection of the elements, which contain a substring specified as a parameter. Since a structure of the document-centric documents is not regular, languages were necessary to extend to operators which provide a querying data with *mixed content*. Consequently, such systems extend classical *information retrieval* (*IR*) models for querying XML data. For example, a query retrieving the elements containing an ordered term or phrase are required in the case of querying document-centric XML documents. In Chapter 2 some existing query languages and index approaches for part-match querying XML data are described.

This paper addresses the indexing and querying the document-centric XML documents using multi-dimensional data structures. Chapter 3 presents previously published multi-dimensional approach to indexing data-centric XML documents, particularly how to map paths to points in a multi-dimensional space. Our approach enables an efficient accomplishment of querying text content of an element or an attribute value as well as of queries based on regular path expressions and XPath axes. Chapter 4 extends this approach for part-match querying XML data. Chapter 5 describes multi-dimensional data structures *UB-tree* [2] and *R^*-tree* [3], which are used for indexing XML document. In conclusion we summarize the paper content and outline possibilities of a future work.

2 State of the art

Recently there are many languages and algorithms deal with matching phrases in XML documents with a mixed content. Now, some of them are described. The XQuery-IR [7] is an extension of the XQuery language which supports phrase matching in document fragments and ranks them according to their relevance by using $TF \times IDF$ weights. A tenet of this weight consists in preference terms that occur frequently with one fragment and infrequently in the rest of document. The XXL [19] is a system with similar syntax like SQL language and using a part-match operator allowed querying conformable terms contained in the element or attribute name. The XIRQL [9] language exploits weights and vague predicates, using appropriate DTD, and creates disjoint index contexts for $TF \times IDF$ weights. The weights are applied to rank of relevant document parts regarding to a specified query.

The XKeyword [12] system applies the rank based on a graph distance between the matched words and allows matching words anywhere in a document. Algorithm *PIX (Phrase matching In XML)* [2] for phrase and similar phrase matching in XML documents does not need a exact path specification like XPath. This algorithm provides a phrase matching overlapping separate elements. The *TIX (Text In XML)* algebra [1] uses the scored pattern tree which contains formulas of boolean combination of predicates (applicable to nodes), a set of scoring function (calculate of the score for each node) and also edges labelled in the sense of XPath axis. Operators as the selection, projection, join, and so on are defined under TIX algebra and enable a ranking relevant elements which contain the phrase in dependence on a document structure.

3 Multi-dimensional Approach to Indexing XML Data

In [13] a multi-dimensional approach to indexing XML data was introduced. A revision of this approach was described in [14]. This approach applies multi-dimensional data structures (see Section 5) to indexing XML data.

3.1 Model of XML documents

An XML document may be modelled by a tree, whose nodes correspond to elements and attributes. String values of elements or attributes or empty values occur in leafs. An attribute is modelled as a child of the related element. Consequently, an XML document may be modelled as a set of paths from the root node to all leaf nodes. Note, unique number $id_U(u_i)$ of a node u_i (element or attribute) is obtained by counter increments according to the document order [11]. Unique numbers may be obtained using an arbitrary numbering schema. Of course, the document order must be preserved.

Let $\mathcal{P}$ be a set of all paths in a XML tree. The path $p \in \mathcal{P}$ in an XML tree is sequence $id_U(u_0), id_U(u_1), \ldots, id_U(u_{\tau_P(p)-1}), s$, where $\tau_P(p)$ is the length of the path p, s is `PCDATA` or `CDATA` string, $id_U(u_i) \in D = \{0, 1, \ldots, 2^{\tau_D} - 1\}$, τ_D is the chosen length of binary representation of a number from domain D. Node u_0 is always the root node of the XML tree. Since each attribute is modelled as a super-leaf node with `CDATA` value, nodes $u_0, u_1, \ldots, u_{\tau_P(p)-2}$ represent elements always.

A labelled path lp for a path p is a sequence $s_0, s_1, \ldots, s_{\tau_{LP}(lp)}$ of names of elements or attributes, where $\tau_{LP}(lp)$ is the length of the labelled path lp, and s_i

```
<!DOCTYPE books [
  <!ELEMENT books(book)>
  <!ELEMENT book(title,author)>
  <!ATTLIST book id CDATA #REQUIRED>
  <!ELEMENT title(#PCDATA)>
  <!ELEMENT author(#PCDATA)>
]>
```

```
<?xml version="1.0" ?>
<books>
 <book id="003-04312">
  <title>The Two Towers</title>
  <author>J.R.R. Tolkien</author>
 </book>
 <book id="001-00863">
  <title>The Return of the King</title>
  <author>J.R.R. Tolkien</author>
 </book>
 <book id="045-00012">
  <title>Catch 22</title>
  <author>Joseph Heller</author>
 </book>
</books>
```

Fig. 1. (a) DTD of documents which contain information about books and authors. (b) Well-formed XML document valid w.r.t. DTD.

is the name of the element or attribute belonging to the node u_i. Let us denote the set of all labelled paths by $\mathcal{LP}$. A single labelled path belongs to a path, one or more paths belong to a single labelled path. If the element or attribute is empty, then $\tau_P(p) = \tau_{LP}(lp)$, else $\tau_P(p) = \tau_{LP}(lp) + 1$.

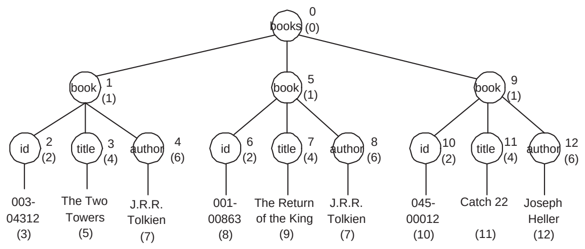

Fig. 2. Example of XML tree with unique numbers $id_U(u_i)$ of elements and attributes u_i and unique numbers $id_T(s_i)$ of names of elements and attributes and their values s_i (values in parenthesis).

Example 1 (Decomposition of XML tree to paths and labelled paths).
In Figure 1 we see an example of an XML document. In Figure 2 we see an XML tree modelling the XML document. We see that this XML document contains paths:

– `0,1,2,'003-04312'`; `0,5,6,'001-00863'` ; and `0,9,10,'045-00012'` belong to the labelled path `books,book,id`,

– `0,1,3,'The Two Towers'`; `0,5,7,'The Return of the King'`; and `0,9,11, 'Catch 22'` belong to the labelled path `books,book,title`,

– `0,1,4,'J.R.R. Tolkien'`; `0,5,8,'J.R.R. Tolkien'`; and `0,9,12,'Joseph Heller'` belong to the labelled path `books,book,author`.

The *term index* which contains all strings s_i of an XML document and their unique numbers $id_T(s_i)$ is used in this approach.

Definition 1 (point of n-dimensional space representing a labelled path).
Let $\Omega_{LP} = D^n$ be an n-dimensional space of labelled paths, $|D| = 2^{\tau_D}$, and $lp \in \mathcal{LP}$ be a labelled path $s_0, s_1, \ldots, s_{\tau_{LP}(lp)}$, where $n = max(\tau_{LP}(lp), lp \in \mathcal{LP}) + 1$. ***Point of n-dimensional space representing a labelled path*** *is defined $t_{lp} = (id_T(s_0), id_T(s_1), \ldots,\ id_T(s_{\tau_{LP}(lp)})) \in \Omega_{LP}$, where $id_T(s_i)$ is a unique number of term s_i, $id_T(s_i) \in D$. A unique number $id_{LP}(lp_i)$ is assigned to lp_i.* ■

Definition 2 (point of n-dimensional space representing a path).
Let $\Omega_P = D^n$ be an n-dimensional space of paths, $|D| = 2^{\tau_D}$, $p \in \mathcal{P}$ be a path $id_U(u_0), id_U(u_1), \ldots, id_U(u_{\tau_{LP}(lp)}), s$ and lp a relevant labelled path with the unique number $id_{LP}(lp)$, where $n = max(\tau_P(p), p \in \mathcal{P}) + 2$. ***Point of n-dimensional space representing path*** *is defined $t_p = (id_{LP}(lp), id_U(u_0), \ldots,$ $id_U(u_{\tau_{LP}(lp)}),\ id_T(s)) \in \Omega_P$.* ■

We define three indexes:

1. **Term index**. This index contains a unique number $id_T(s_i)$ for each term s_i (names and text values of elements and attributes). The unique numbers can be generated by counter increments according to the document order. We want to get a unique number for a term and a term for a unique number as well. This index can be implemented by the B-tree.
   In Figure 2 we see the XML tree with unique numbers of terms in parenthesis.
2. **Labelled path index**. Points representing labelled paths together with labelled paths' unique numbers (also generated by counter increments) are stored in the labelled path index.
   In Figure 2 we see that the document contains three unique labelled paths `books,book,id`; `books,book,title`; and `books,book,author`. We create points `(0,1,2)`; `(0,1,4)`; and `(0,1,6)` using id_T of element's and attribute's names. These points are inserted into a multi-dimensional data structure with id_{LP} `0`, `1`, and `2`.
3. **Path index**. Points representing paths are stored in the path index.
   In Figure 2 we see unique numbers of elements. Let us take the path to the value `The Two Towers`. Relevant labelled path `books,book,title` has got id_{LP} 1 (see labelled path index). We get point `(1,0,1,3,5)` after inserting unique numbers of labelled path id_{LP}, unique numbers of elements id_U and term `The Two Towers`. This point is stored in a multi-dimensional data structure.

An XML document is transformed to points of vector spaces and XML queries are implemented using a multi-dimensional data structure queries. The multi-dimensional data structures provide a nature processing of *point* or *range queries* [2]. The point query probes if the vector is or is not present in the data structure. The range query searches all points in a query box $T_1 : T_2$ defined by two points T_1, T_2.

3.2 Queries for values of elements and attributes

Now, implementation of a query for values of elements and attributes and query defined by a simple path based on an ancestor-descendent relation will be described. Query processing is performed in three phases which are connected:

1. **Finding unique numbers id_T of query's term in the term index**.
2. **Finding labelled paths' id_{LP} of query in the labelled path index**. We search the unique numbers in a multi-dimensional data structure using point or range queries.
3. **Finding points in the path index**. We find points representing paths in this index using range queries. Now, we often want to retrieve (using labelled paths and term index) names or values of elements and attributes.

4 Efficient part-match querying of XML data

Now, an extension of the multi-dimensional approach for part-match querying XML data is described. We aim to querying individual terms of the element and attribute string values mainly. Operator $\sim=$ is defined for such query. The individual terms must be indexed, but we need preserve an information about pertinence of the term to the path and labelled path. The *Path-Labelled path-Term* (*PLT*) index satisfies such requirements. This storage contains points of an 3-dimensional space $\Omega_{PLT} = D_{id_P} \times D_{id_{LP}} \times D_{id_T}$. Consequently, items of the space are points $(id_P(p_i), id_{LP}(lp_i),\ id_T(t_i))$. In order to the index can be used for a part-match querying, a unique number $id_P(p_i)$ of path p_i is stored in the first coordinate of the point representing the path p_i: $t_{p_i} = (id_P(p_i), id_{LP}(lp_i), id_U(u_0)$, $id_U(u_1), \ldots,\ id_U(u_{\tau_P(p_i)}))$. During a parsing string values of elements and attributes we could use the stop-list known in IR systems [3]. For example, frequent terms (e.g. conjunctions) are eliminated by the stop-list. Such terms are not important for a querying. Since whole values of elements and attributes are important and $it_T(t_i)$ of the whole string t_i is removed in the point representing the path, whole short values are inserted in the term index and PLT index.

Example 2 (Creation of the PLT index).

Let us take a document-centric XML document. For example, Shakespeare's Hamlet in XML [6]. $id_{LP}('PLAY, SCENE, ACT, SPEECH, SPEAKER') =$ 100 and id_P of belonging path is 110 (see Figure 3).

```
<PLAY>...
  <SCENE>...
    <ACT>...
      <SPEECH>
        <SPEAKER>MARCELLUS</SPEAKER>
        <LINE>It faded on the crowing of the cock.</LINE> ...
      </SPEECH>
      ...
```

Fig. 3. A part of Shakespeare's Hamlet in XML.

$id_{LP}('PLAY, SCENE, ACT, SPEECH, LINE') = 101$ and id_P of belonging path is 111. Unique numbers of terms: $id_T('MARCELLUS') = 120$, $id_T('crowing') = 121$, $id_T('cock') = 122$, and so on. After the insertion of points representing the path, labelled path, $id_T(t_i)$ and term t_i into the term index, the points are created and inserted into the PLT index: $(110, 100, 120)$, $(111, 101, 121)$, $(111, 101, 122)$, and so on. ■

Now, processing the query `/books/book[keywords∼='XML']/title` over an XML database of books is described. Note, query box $(qb_1, min(D), \ldots, min(D)) : (qb_1, max(D), \ldots, max(D))$ may be written as $(qb_1, *, \ldots, *)$.

1. Finding $id^1_{LP} = id_{LP}('books, book, keywords')$ and $id^1_T = id_T('XML')$.
2. Processing the narrow range query $(*, id^1_{LP}, id^1_T)$ in the PLT index. The result is k unique numbers $id_P(p_1), \ldots, id_P(p_k)$ of relevant paths $p_1, \ldots, p_k$.
3. Processing the complex range query $(id_P(p_1), id^1_{LP}, *, \ldots, *)$,...,$(id_P(p_k), id^1_{LP}, *, \ldots, *)$. The result is the points representing the relevant paths.
4. Finding $id^2_{LP} = id_{LP}('books, book, title')$.
5. Performing the child XPath axis with id^2_{LP} in the second coordinate. The child XPath axis is implemented by a sequence of range queries (see [14]). The result is m paths $p^f_1, \ldots, p^f_m$.
6. Performing the complex range query $(id_P(p^f_1), id^2_{LP}, *), \ldots, (id_P(p^f_m), id^2_{LP}, *)$. An output is collection of $id_T(t_i)$. Strings of titles t_i are retrieved from the term index and the strings are returned as a result.

Query processing of a general part-match query is a generalization of above described procedure. The XML query languages make it possible to place a complex query condition using boolean operators e.g., `AND` and `OR`. In described approach a query defined by the `OR` operator is possible to process effectively. For example, the query `/books/book[keywords∼='XML' OR keywords∼='SGML']/title` is performed according to above techniques, but the first two steps are distinguish.

1. Finding $id^1_{LP} = id_{LP}('books, book, keywords')$, $id^1_T = id_T('XML')$, and $id^2_T = id_T('SGML')$.

2. Processing the narrow range queries $(*, id^1_{LP}, id^1_T)$ and $(*, id^1_{LP}, id^2_T)$ in the PLT index. The result is k unique numbers $id_P(p_1), \ldots, id_P(p_k)$ of relevant paths $p_1, \ldots, p_k$.

Next steps of this query processing are the same.

5 Index Data Structures

Due to the fact that an XML document is represented as a set of points representing paths and labelled paths in the multi-dimensional approach, we use multi-dimensional data structures for their indexing, e.g., paged and balanced multi-dimensional data structures like UB-tree [2], and R*-tree [3].

(B)UB-tree data structure applies *Z-addresses* (*Z-ordering*) [2] for mapping a multi-dimensional space into single-dimensional. Intervals on *Z-curve* (which is defined by this ordering) are called *Z-regions.* (B)UB-tree stores points of each Z-regions on one disk page (tree leaf) and a hierarchy of Z-regions forms an index (inner nodes of tree). In the case of indexing point data, an R-tree and its variants cluster points into *minimal bounding boxes* (*MBB*s). Leafs contain indexed points, super-leaf nodes include definition of MBBs and the other inner nodes contain hierarchy of MBBs. (B)UB-tree and R-tree support *point* and *range queries* [11], which are used in the multi-dimensional approach to indexing XML data. The range query is processed by iterating through the tree and filtering of irrelevant tree nodes, i.e. (super)Z-regions in the case of (B)UB-tree and MBBs in the case of R-tree, which do not intersect a query box.

One more important problem of the multi-dimensional approach is the unclear dimension of spaces of paths and labelled paths. A naive approach is to align the dimension of space to the maximal length of path. For example, points of dimension 5 will be aligned to dimension 36. This technique increases the size of index and the overhead of data structure as well. In [15] BUB-forest data structure was published. This data structure solves the problem of indexing points with different dimensions. The range query used in the multi-dimensional approach is called *narrow range query.* Points defining a query box have got some coordinates the same, whereas the size of interval defined by other coordinates near to the size of space's domain. Many irrelevant regions are searched during processing the narrow range query in multi-dimensional data structures. In [9] Signature R-tree data structure was introduced. This data structure enables efficient processing the narrow range query.

6 Conclusion

In our future work we would like to test this approach over a current test XML document collections. Test queries are often defined for such collections. Therefore a comparison of our approach with another XML indexing approaches is possible. INEX [10] collection seems to be hopeful. INEX contains 12,000 IEEE articles since 1995. The size of the collection is 500MB.

References

1. S. Al-Khalifa, C. Yu, and H. Jagadish. Querying Structured Text in an XML Database. In *Proceedings of International Conference on Management of Data (SIGMOD), San Diego, CA*, June 2003.
2. S. Amer-Yahia, M. Fenández, D. Srivastava, and Y. Xu. Phrase Matching in XML. In *Proceedings of the 29th VLDB Conference, Berlin, Germany*, 2003.
3. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison Wesley, New York, 1999.
4. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of WWCA'97, Tsukuba, Japan*, 1997.
5. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331.
6. J. Bosak. Shakespeare in XML, 1999, `http://www.ibiblio.org/xml/examples/shakespeare/`.
7. J.-M. Bremer and M. Gertz. XQuery/IR: Integrating XML document and data retrieval. In *Proceedings of the 5th International Workshop on the Web and Databases (WebDB)*, June 2002.
8. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. Technical report, WWW Consortium, August, 1998.
9. N. Fuhr and K. Grossjohann. XIRQL: An extension of XQL for information retrieval. In *Proceedings of SIGIR*, 2001.
10. N. Fuhr, N. Gvert, S. Malik, M. Lalmas, and G. Kazai. INEX – Initiative for the Evaluation of XML Retrieval, 2003, `http://www.is.informatik.uni-duisburg.de/projects/inex/index.html.en`.
11. T. Grust. Accelerating XPath Location Steps. In *Proceedings of ACM SIGMOD 2002, Madison, USA*, June 4-6, 2002.
12. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In *Proceedings of the ICDE*, 2003.
13. M. Krátký, J. Pokorný, T. Skopal, and V. Snášel. The Geometric Framework for Exact and Similarity Querying XML Data. In *Proceedings of First EurAsian Conferences, EurAsia-ICT 2002, Shiraz, Iran.* Springer–Verlag, LNCS 2510, 2002.
14. M. Krátký, J. Pokorný, and V. Snášel. Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In *Accepted at International Workshop DataX, Int'l Conference on EDBT, Heraklion - Crete, Greece*, 2004.
15. M. Krátký, T. Skopal, and V. Snášel. Multidimensional Term Indexing for Efficient Processing of Complex Queries. *Kybernetika, Journal of the Academy of Sciences of the Czech Republic, accepted*, 2003.
16. M. Krátký, V. Snášel, J. Pokorný, P. Zezula, and T. Skopal. Efficient Processing of Narrow Range Queries in the R-Tree. In *Submitten at VLDB 2004*, 2004.
17. Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of 27th VLDB International Conference*, 2001.
18. J. Pokorný. *XML: a challenge for databases?*, pages 147–164. Kluwer Academic Publishers, Boston, 2001.

19. A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *Proceedings of EDBT*, 2002.
20. W3 Consortium. XQuery 1.0: An XML Query Language, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath/`.
21. W3 Consortium. XML Path Language (XPath) Version 2.0, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath20/`.
22. W3 Consortium. Extensible Markup Language (XML) 1.0, 1998, `http://www.w3.org /TR/REC-xml`.
23. W3 Consortium. XML Schema Part 1: Structure, 2001, `http://www.w3.org/TR/xmlschema-1/`.
24. C. Yu. *High-Dimensional Indexing.* Springer–Verlag, LNCS 2341, 2002.