XML Query Processing Using Signature and DTD*

Sangwon Park[1], Yoonra Choi[2], and Hyoung-Joo Kim[3]

[1] Department of Digital Contents, Sejong Cyber University, Seoul, Korea
swpark@cybersejong.ac.kr
[2] Korea Computer Communications, Ltd., Seoul, Korea
yulla@unisql.com
[3] School of Computer Science and Engineering, Seoul National University, Seoul, Korea
hjk@oopsla.snu.ac.kr

Abstract. Having emerged as a standard web language, XML has become the core of e-business solution. XML is a semistructured data that is represented as graph, which is a distinctive feature compared to other data dealt with existing database. And query is represented as regular path expression, which is evaluated by traversing each node of the graph. In XML document with DTD, the DTD may be able to provide many valuable hints on query optimization, because it has information on the structure of the document. Using signature and information from DTD, we can minimize the traverse of nodes and quickly execute the XML query of regular path expression fast.

1 Introduction

One of the most attractive technologies in the 21st century would be XML. Various kinds of standards have been established, and much more data will be represented by XML. XML is very similar to semistructured data[1,2] which is based on the OEM model.

The query on semistructured data often contains regular path expression which requires much more flexible query processing than the ones for relational or object oriented databases for retrieving irregular and sometimes unknown structure. Several indexing methods for semistructured data are proposed[6,8, 9].

The signature method has been proposed to process queries[3,5,11,14]. This method reduces the search space according to result value from bit operations of hash values. When an XML data is stored in object repositories and each node is stored as an object, the signature method can be used, which decreases the number of fetching nodes during query processing[10]. However this method has a problem that the signature bits can be saturated when the number of sub elements or the depth of the tree is increased. And at the same time, there

* This work was supported by the Brain Korea 21 Project.

K. Bauknecht, A M. Tjoa, G. Quirchmayr (Eds.): EC-Web 2002, LNCS 2455, pp. 162–171, 2002.

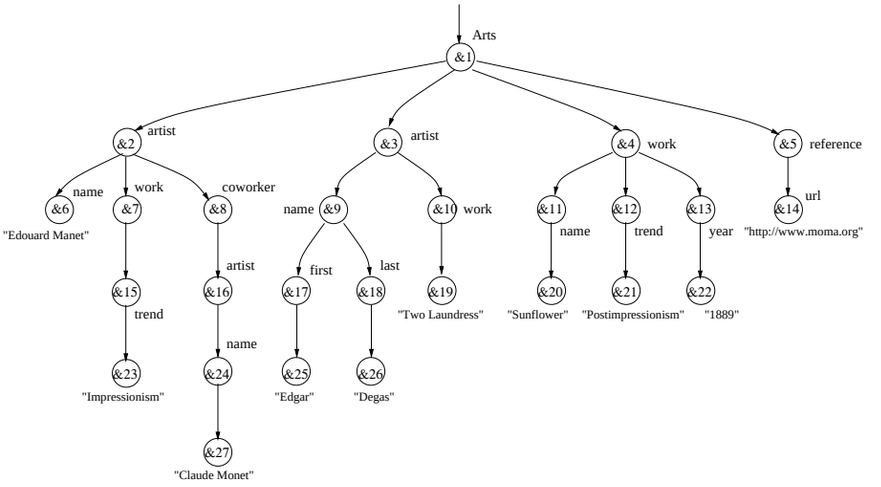


Fig. 1. DOM graph

has been studies on query processing using DTD information which has the structural hints as to an XML document[13].

In this paper we propose to improve the performance of query processing using both signature and DTD which are helpful to reduce the search space. To do this, we first find all possible paths suitable for the given query in DTD declaration and merge them into one query graph. Whenever visiting each node in DOM, we compare between the signature of a node in the data graph and the query graph. It reduces the search space and minimize the number of fetching node in object repositories.

The remainder of this paper is organized as follows: Section 2 presents related works. Section 3 briefly explains what a signature is. The sequential steps of query optimization using signature and DTD are given in Section 4 and 5. And Section 6 discuss the experimental results. Finally, conclusions and future works are presented in Section 7.

2 Related Work

The query on XML often contains regular path expressions. This means the expression of a query contains wild cards like *, $, and so on. DataGuide[6] can decrease query processing time by the path index which gives a dynamic outline of the structure of semistructured data. But this is only applicable to the query with single regular expression and not applicable to the query with complex regular expression having various ones. The 2-Index[9] overcomes the weak point of previous work, but the size of indexes may be the square of the data nodes. T-Index partially solved the problems in 2-Index, but still could not overcome the problem where indexes can not cover all possible paths.

The query processing techniques using signature also have been largely studied [3,5,14,10]. In relational database, the signature technique is used to select matched tuples by select condition[3]. The signature is used in object-oriented database to reduce page I/O when evaluate the path expression[14]. However, regular path expressions can not be processed by these methods.

Lore[7] and eXcelon[4] are the representative databases that deal with semistructured data or XML. These keep the structure of graph for atypical data and store the nodes of the graph as objects. As the query is processed by visiting nodes of a tree, the reduction of the node fetching is the key point for query optimization.

3 Preview on Signature

XML data can be depicted as a tree such as Figure 1. If each node in this graph is stored as an object in object repository, the objects have to be visited when evaluating regular path expression.

The method of generating a signature is the similar to the one of [5] where a signature is built from hash values of element labels. Each node of DOM tree is stored as an object with a signature. The signature of each node is hash value from label of the matching element. An upper node contains signature information of the lower nodes and the signature is generated by bit-wise ORing of the signatures of lower nodes. Let the hash value of a node n_i be H_j, the signature be S_i, and the child node of n_i be n_j. On this assumption, it is true that $S_i = H_i \vee (\bigvee_j S_j)$. For example, the signature value of node &9 in Figure 1 is the result of bit-wise ORing of the signatures of nodes &17, &18, and the hash values to names of the nodes.

An example for traversing nodes of DOM tree using signature is as follows. For node &2 in Figure 1, the bit-wise AND of signatures results in like

$$H_{\text{“}coworker\text{”}} \wedge S_{\&2} \equiv H_{\text{“}coworker\text{”}}$$

So it is possible that the node with label of `coworker` exists in the sub-tree of node &2. But in the case of node &4, the result is

$$H_{\text{“}coworker\text{”}} \wedge S_{\&4} \neq H_{\text{“}coworker\text{”}}$$

In this case, we can be sure that there is no node having the label, `coworker`, in the sub-tree, and thus we need not visit the child nodes of &4 [10].

4 Building DOM Based on Signature

As we have seen in Section 3, we traverse the sub-tree of a node only when the value of bit-wise AND of signature of the visited node and hash value of the required element is equal to the hash value. Otherwise, the sub-tree are not visited, which would reduce the search space and eventually speed up query processing.

```
<!ELEMENT Arts (artist+,work+)>
<!ELEMENT artist (name,work,coworker?>
<!ATTLIST artist name CDATA \#IMPLIED>
<!ELEMENT work (#PCDATA | name | trend | year)*>
<!ELEMENT reference (url)>
<!ATTLIST reference url CDATA #IMPLIED>
<!ELEMENT coworker (artist)>
<!ELEMENT name (#PCDATA | (first, last))>
<!ELEMENT trend (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last (#PCDATA)>
```

Fig. 2. DTD

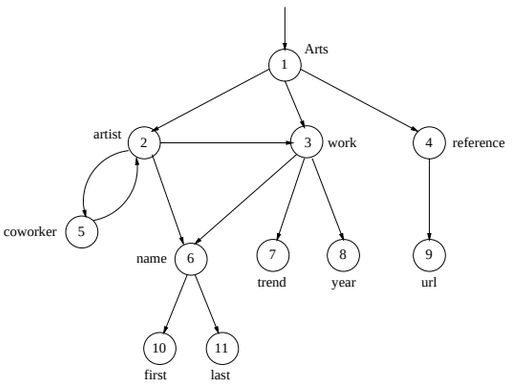


Fig. 3. DTD graph

However, if there is more elements or if the sub-tree is deeper, it is more likely that all bits of signature are set to '1'. We call thus saturation. If saturation occurs in node n_i, the equation $H_l \wedge S_i \equiv H_l$ is true even if the label l does not exist in the sub-tree of node n_i. This problem can be solved by extracting only indeterminate elements.

4.1 Extraction of Indeterminate Elements

Unlike [10], this paper does not obtain hash values from all elements. Figure 2 is DTD of the Figure 1. The DTD can be depicted as Figure 3. Following the definition in DTD, each element can be separated into either determinate or indeterminate one. It can be judged by the definition of sub-element of an element. There is a quantity indicator attached to the sub-element such as '?', '*', or '+'. The mark '?' or '*' means that this element may not appear in the XML document. While '+' means that this element must appear in the XML

Table 1. Hash values of the labels

Arts	00000000	first	11000110	artist	00000000	last	11011000
work	00000000	trend	11111100	reference	00000000	year	10001111
coworker	11001110	url	00000000	name	11000110		

Table 2. Signatures of each Node

&1	11111111	&2	11111110	&3	11111110	&4	11111111	&5	00000000
&6	11000110	&7	11111110	&8	11001110	&9	11011110	&10	00000000
&11	11000110	&12	11111100	&13	10001111	&14	00000000	&15	11111100
&16	11001110	&17	11000110	&18	11011000	&24	11000110		

document once or more time. The former is an indeterminate element, while the latter is called a determinate one.

And a sequence connector of sub-elements may be ',' or '|'. The connector ',' means that the elements will be enumerated in order. On the other hand, '|' means that only one of the elements connected by '|' may appear but it can not be known in advance which one will appear in the actual XML document. Therefore, the elements connected by ',' are determinate, while the ones connected by '|' are indeterminate.

Definition 1 *The element with quantity indicator is '?' or '*' and the elements connected by '|' are defined as indeterminate. All the other elements are defined as determinate.*

4.2 Generating DOM Based on Signature

It is assumed that each node in DOM tree is stored as an object with a signature. The signature of each node is evaluated by bit-wise ORing from hash values and signatures of child nodes. The process for calculating the signature of a node follows Algorithm 1.

5 Query Optimization Using a Query Graph

5.1 Generating Query Graph

We extract all paths satisfying the given query in advance, which means that we are filtering the sequences of labels to visit. We visit only the nodes on the paths which limit the scope of search. This section will explain how all paths for the given query is extracted.

Example 1 *The NFA of regular path expression* `Arts.*.name` *is as follows. This is generated by the rules defined in [10].*

Algorithm 1 MakeSignature(node)

```
1: s ← 0
2: if node is an Element or Attribute node then
3:    for each ChildNode of node do
4:       s ← s ∨ MakeSignature(ChildNode) /* bit-wise ORing */
5:       if node is indeterminate then
6:          s ← s ∨ hash value of node
7:       end if
8:    end for
9: end if
10: node.signature ← s
```

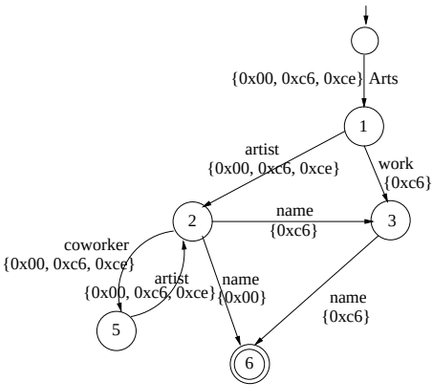
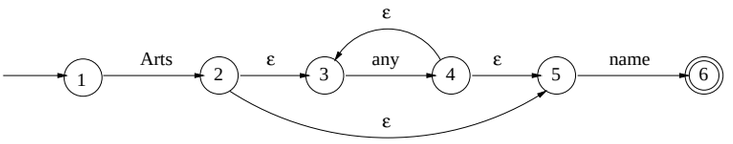


Fig. 4. Query Graph



We transform the regular path expression into NFA. Example 1 shows the NFA of a regular path expression. Then we traverse the DOM graph such as Figure 3 to get a query graph. Figure 4 is the query graph of the query in Example 1. The labels of the query graph is different from the DTD graph. The label of a node is attached to in-coming edges. The signature of the query graph is made by propagating the hash value of the indeterminate label from final node to the start node. The detailed algorithm is described in [12].

5.2 Query Optimization Using a Query Graph

The signature of each node in the DOM tells which labels exist in the sub-tree. And the signature list in the query graph represents the summary of required labels satisfying the given query. Especially in the case of signature list, only

indeterminate elements are considered. By comparing the label and the signature of the node in DOM and the edge in query graph, we determine whether to traverse the sub-tree of the node or not. If the labels are not the same or even if the signature of the node of DOM is not matched with any others in the signature list of the edge in query graph, we will not traverse the sub-tree. Algorithm 2 is the scan operator which returns the nodes satisfying the query in DOM tree.

The function `ForwardLabel()` first compares the labels between the DOM and the query graph. If they are the same, then it compares the signatures between them. This means that we reduce the search space by filtering the nodes through the query paths first and comparing the signatures next.

Algorithm 2 next()

```
1: /* states: state set of QueryGraph */
2: node ← next node by DFS from DOM
3: while node is not NULL do
4:    forwardLabel(states,node)
5:    if existFinal(states) then
6:       return node
7:    end if
8:    if isReject(states) then
9:       node ← next node by DFS from DOM
10:   end if
11: end while
```

Example 2 *If the nodes in the DOM of Figure 1 have the signatures of Table 2, the node fetching process for query of Example 1 using query graph is as follows.*

First the node `&1` *is read, then the state set* $S=\{2\}$*. As the nodes are visited in the DFS order, first child* `&2` *is read and* $S=\{3\}$*. Again the first child* `&6` *is read and* $S=\{4\}$*, where query graph arrives the final state. So the node* `&6` *is returned. The state transition occurs only when the label and the signature of the node is equal to those of the edge of query graph.*

The sibling of node `&6` *is node* `&7`*, where* $S=\{5\}$ *and the label to appear in the query graph is* `name`*. But not even if child of* `&7` *do not have the label* `name` *at all, therefore we do not traverse the sub-tree of* `&7`*. Instead* `&8`*, the sibling node of* `&7`*, is read. By doing this repeatedly, the nodes* `&6, &24, &9` *and* `&11` *are returned in sequence as a result.*

6 Experimental Results

In this section, we will analyze the experimental results of two methods of query optimization, i.e.,the one using signatures only[10] and the other, proposed in this paper, using signature and DTD together. As can be seen from the comparison

Table 3. Characteristics of the XML Files

	No. of Nodes	File Size
Shakespeare	537,621	7.5 Mbytes
Bibliography	19,854	247 Kbytes
The Book of Mormon	142,751	6.7 Mbytes

Table 4. Queries Used in the Experiment

Q1	Shakespeare	PLAY//STAGEDIR
Q2	Shakespeare	//SPEECH//STAGEDIR
Q3	The Book of Mormon	tstmt//p
Q4	The Book of Mormon	//sura/epigraph
Q5	Bibliography	bibliography//'in.*'/year
Q6	Bibliography	//misc

results of the number of node fetching and the number of page I/O, the latter method shows the better efficiency than the first.

6.1 Conditions of Experiment

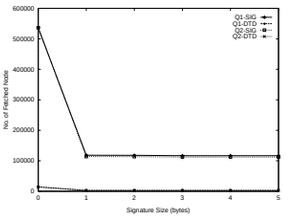
The programs are coded by JAVA, and the data used in this experiment are Shakespeare, The Book of Mormon, and Michael Lay's bibliography. Six queries are used and are described in Table 4. The first queries for each document retrieve the nodes which are located on the specific path. And the second queries retrieve the nodes located at arbitrary positions of DOM tree. The signature technique for query optimization of [10] is targeted to compare with the technique of this paper.

6.2 Performance Evaluation

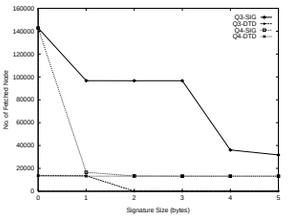
In Figure 5, the vertical axis represents the number of node fetching and the horizontal axis represents the size of signature. A zero sized signature means the method not using signature.

In Figure 5 (a), the number of fetching nodes has decreased to 1/40. In the case of Q3 at Figure 5 (b), the number of fetching nodes has decreased 1/10 and 1/7 times for 0 and 1 byte of signature, respectively, and 1/1660 for two or more bytes. And for Q4, the number of fetching nodes decreased to 1/10 times as small as the one of [10] for zero size signature, and the number of fetching nodes converges for one more bytes of signature. Even in (c), the number of fetching nodes decreased to half in Q5 and decreased from half to less converging point in Q6.

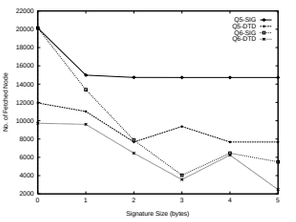
As we have seen from the results in Figure 5, the query optimization using both signature and DTD greatly decreases the number of fetching nodes. Especially the number of fetching nodes is dramatically dropped for one or two bytes



(a) Q1, Q2



(b) Q3, Q4



(c) Q5, Q6

Fig. 5. The number of fetching nodes

of signature, and maintained for more bytes of signature. This is a very strong point that the performance of less bytes signature is equivalent to larger ones, because we are able to decrease the bytes of signature.

The results show that the performance improvement of query optimization is better for the graph of long depth, but is less for the graph of wide breath. This is related to the fact that the deeper the graph, the greater the probability of signature saturation to occur. Bibliography is the data with the structure of slight depth and wide breath. Considering the results of Q3 and Q4, the difference in the number of fetching nodes fetching between the method of [10] and the one proposed in this paper is less than the difference in Q1 and Q2 for Shakespeare.

And as in Q3, if the regular path expression like '//' is not contained in the path expression of query, the number of fetching nodes largely decreases. This is because the query path in DTD is determined uniquely.

7 Conclusion

In the previous sections, we have proposed the technique of XML query optimization using both signature and DTD. By processing the regular path expression with query graph, the numbers of fetching nodes. largely decreased. Especially it shows greater efficiency improvement with even smaller size of signature than the one of [10]. This eventually means that the small size of storage for a node is needed.

Contrary to the efficiency improvement in the selection query, the insertion and deletion queries cause propagation overheads to change signatures of parent nodes. In the latter case, the cost usually exceeds the one of selection query. But most queries on XML document are selections, therefore this paper do not consider the modification queries. The process of changing the signatures of each node in DOM is like the algorithm in [10].

The DTD given by XML v1.0 is insufficient in the respect of providing the accurate and comprehensive information on the structure of XML document. XML schema is the definition format that can overcome such defects in DTD. Thus, the performance of query optimization may be improved by using XML schema and research on it should be considered as future work.

References

1. Serge Abiteboul. Querying Semistructured Data. *International Conference on Database Theory*, January 1997.
2. P. Buneman. Semistructured Data. *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 1997.
3. Walter W. Chang and Hans J. Schek. A Signature Access Method for the Starburst Database System. *VLDB*, 1989.
4. eXcelon. An XML Data Server For Building Enterprise Web Applications. *http://www.odi.com/products/white_papers.html*, 1999.
5. Chris Faloutsos. Signature files: Design and Performance Comparison of Some Signature Extraction Methods. *SIGMOD*, 1985.
6. Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. *VLDB*, 1997.
7. Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3), 9 1997.
8. Jason McHugh and Jennifer Widom. Query Optimization for XML. *VLDB*, 1999.
9. Tova Milo and Dan Suciu. Index Structures for Path Expressions. *ICDT*, 1999.
10. Sangwon Park and Hyoung-Joo Kim. A New Query Processing Technique for XML Based on Signature. *7th International Conference on DASFAA*, pages 22–29, April 2001.
11. R. Sacks-Davis, A. Kent, and K. Ramamohanarao. Multikey Access Methods Based on Superimposed Coding Techniques. *TODS*, 12(4), 1984.
12. Sangwon Park and Yoonra Choi and Hyoung-Joo Kim. XML Query Optimization using Signature and DTD. *Technical report http://swpark.pe.kr/publication.html*, November 2001.
13. Tae-Sun Chung and Hyoung-Joo Kim. Extracting Indexing Information from XML DTDs. *Information Processing Letters*, 81(2), 2002.
14. Hwan-Seung Yong, Sukho Lee, and Hyoung-Joo Kim. Applying Signatures for Forward Traversal Query Processing in Object-Oriented Databases. *ICDE*, 1994.