Indian Journal of Science and Technology Vol 8(32), DOI: 10.17485/ijst/2015/v8i32/92106, Nov 2015

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Node Indexing in XML Query Optimization: A Review

Su-Cheng Haw* and Aisyah Amin

Faculty of Computing and Informatics, Jalan Multimedia, Cyberjaya - 63100, Malaysia;
sucheng@mmu.edu.my, tgaisyah.amin@gmail.com

Abstract

Background/Objectives: Node indexing has been developed to optimize query retrieval. Since its inception in the early century, there are many node indexing techniques. **Methods/Statistical Analysis**: Node indexing can be group into four major groups which is, subtree labeling, prefix-based labeling, multiplicative labeling and hybrid labeling. Each indexing techniques has its advantages and disadvantages. However, there is an absence of literature reviews on the review of the recent techniques; the latest one was in year 2009. As such, this research project aims to review on some of the latest techniques for each node indexing group. **Findings:** Choosing a correct indexing is critical. For example, prefix-based indexing scheme size grows too huge, while high computation cost is needed to annotate using multiplicative labeling. On the other hand, the subtree group is weak in data updates, while a hybrid scheme combining various schemes with the aim to create a scheme with the strengths of several schemes. **Application/Improvements:** Most important, this review explores and identifies the trends which can be useful for new researcher.

Keywords: Labeling Scheme, Node Indexing, Query Optimization, Query Retrieval, XML Database

1. Introduction

eXtensible Markup Language or better known as XML in short, is gaining its popularity since its inception in the early 2000s. XML files are widely used to generate and share common information on World Wide Web and intranet by using standard ASCII text. It is used for human-readable and machine-readable, due to its nature of self-describing. In XML, tags, which are known as element, are enclosed in angle brackets to describe the content it surrounds. For instance, Figure 1 classifies the title for the book with id '0-1-13' is 'XML Overview'. Each XML file contains single root. From Figure 1, publications is the root. Below publications, there are two sub-elements, which consist of two books. Below the first book element, there are two sub-elements consists of title and chapter and so on. The tree representation of Publication List is shown in Figure 2.

Indexing[1-3] is well-established to improve the query processing speed by decreasing the search space. With indexing, the input query is matched against the index tree, which is usually much smaller instead of the document itself.

Since XML is semi-structured data, in addition to the text queries, the support for complex structural queries is crucial[4,5]. A structural search is to retrieve matches on the tree where it has the tags and structure (relationship) specified in the query criteria. There are three main types of relationships; namely, Parent-Child (P-C), Ancestor-Descendant (A-D) and siblings (the order of the node).

Structural indexes are classified into three main groups; Path indexing, Node indexing and Sequence-based indexing. Nevertheless, the focus of this paper is on Node indexing.

Node indexing can be group into four major groups which is, subtree labeling, prefix-based labeling, multiplicative labeling and hybrid labeling[6].

Among all, Subtree is the modest category. The label of a given document node v in D encodes the position and the extent of the subtree Dv of D that is rooted in v. In another words, this means that by taking the offsets in the sequence of nodes, we will be able to compute the label in a specific order. In terms of determination on the structural relationships, for the given nodes v, w, in D,

*Author for correspondence

```
<publications>
  <book id = "0-1-13">
    <title> XML Overview</title>
    <chapter>
     <title> Introduction </title>
     <section>
       <text>Queries</text>
       <figure>
         <caption> XML Overview</caption>
        </figure>
      </section>
     </chapter>
  </book>
    <book id = "0-1-13">
    <title> Indexing</title>
  </book>
</publications>
```

Figure 1. PublicationList: An example of XML document.

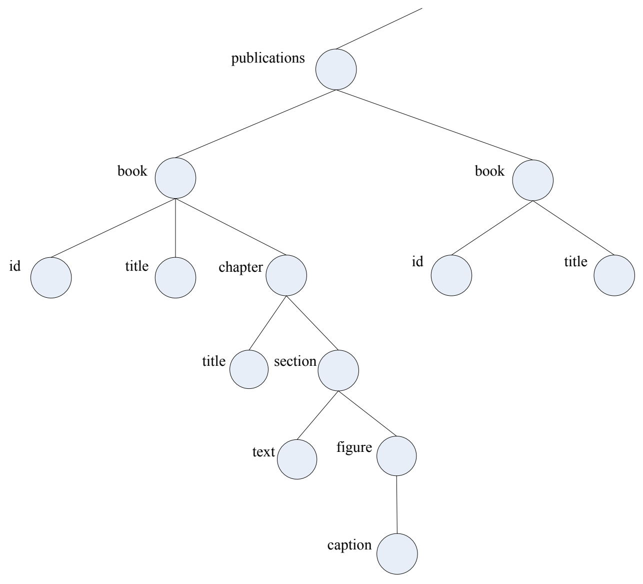


Figure 2. PublicationList in tree representation.

their A-D and P-C relationships is always determined by testing whether Dv contains Dw.

Interval encoding and region encoding are under the Subtree grouping. These encoding technique is described as below:

- In internal encoding, node is labeled with a couple of unique integer assigned by the traversing the tree in preorder and postorder manner.
- The region encoding is known as range encoding schemes recognized by region coordinate. The region coordinate is usually a pair of integers containing the start position, and end position of the substring, from the root of the XML document.

Being the most diverse class of labeling schemes, Prefix-based is also known as path-based labeling scheme. Using this group of labeling scheme, the node v in D usually carries some prefix annotation from the ancestor of the same path. We could easily determine the relationship precisely, i.e., u is an ancestor of v iff label(u) is a prefix of label(v).

Multiplicative Labeling used atomic numbers to identify nodes. This labeling scheme also used some of arithmetic to figure the relationship between nodes.

On the other hand, Hybrid Labeling scheme uses mixed grouping of the strengths of existing methods to support faster query processing.

Some of the recent indexing methods are discussed in the next section.

2. Review on Existing Node Indexing

2.1 Extended Inverted List (Interval encoding)

Extended Inverted List[7] is an example of Interval encoding. Basically, this labeling scheme uses the Nested Tree structure to support dynamic update. In their definition[7], a Nested Tree is a subtree which has an

- interval-based number as a node of the containing tree and
- Its own interval-based numbering as a tree.

Using this labeling scheme, each node is represented as 4-tuple (DocID, sList, eList, Level), where by DocID is the identifier of the document, sList and eList is the startList and endList of the node respectively, and Level is the depth of the node in the XML tree. The startList of any tree node N is the list, s1,...,sn;sn+1, where the last Nested Tree T of N is an nNested Tree, where si is the label of the i-Nested Tree of the node N(i = 1, 2,...,n) and sn+1 is the start position of N in the n-Nested Tree T. The endList of node N is assigned similar to the startList of N except that we used the end position instead. Figure 3 shows the XML tree labelled with Extended Inverted List (except DocID and Level).

Using this property, the relationships among the nodes can be determined as illustrated in the 3 cases below.

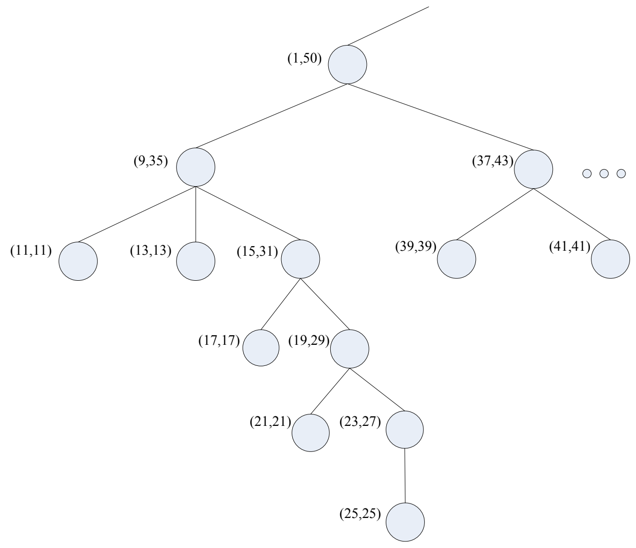


Figure 3. Extended Inverted List.

Case 1:

"If m = n = k, in other words the last Nested Tree of N1 and N2 is (k 1)-Nested Tree, by the definition of startList and endList, sk and ek are the start position and end position of N1 in the (k 1)-Nested Tree, and tk and fk are the start position and end position of N2 in the same (k 1)-Nested Tree. Therefore N1 is the ancestor of N2 if and only if sk < tk and fk < ek from the property of the interval-based numbering."

Example: Let us assume that N1 is node (9, 35) and N2 is node (19,29). According to the theorem, N1 is the ancestor of N2 if and only if 9 < 19 (in this case, it is true) and 29 <35 (in this case, it is true). As such, N1 and N2 is having A-D relationship.

Case 2:

"If min(m,n) > k, N1 and N2 are included in kNested Trees, and the k-Nested Tree of N1 is different from that of N2. Therefore, there is not an ancestor–descendant relationship between N1 and N2. Since, sk = ek and tk = fk in this case, sk < tk and fk < ek is false. Consequently, the theorem holds for this case."

Example: Let N1 be node (39,39) and N2 be node (41,41). To check whether the two nodes are in A-D relationship, 39 < 41 (in this case is true), but 41 < 39 (in this case is false). As such, the two nodes does not have A-D relationship.

Case 3:

"If min (m,n) = k(suppose m = k), in other words the last Nested Tree of N1 is (k 1)-Nested Tree and that of N2 is not, then tk = fk and tk is the label of the k-Nested Tree of N2. sk and ek are the start position and end position of N1 in the (k 1)-Nested Tree. Therefore, sk < tk < ek if and only if the k-Nested Tree of N2 is a descendant of N1, in other words, N2 is a descendant of N1."

Example: Let N1 be node (9,35) and N2 be node (23,27). To check whether the two nodes are in A-D relationship, we need to figure out whether the following is true: 9 < 23 < 35 (in this case is true). As such, the two nodes are in A-D relationship.

The advantage of this labeling scheme is it also supports dynamic update, i.e., the insertion and deletion processing take place with almost no node re-labeling required. The process of insertion XML data is done by adding a subtree into the XML tree. They proposed an Insert algorithm to handle the space at the position of the insertions, and labeling nodes in the inserted subtree with integer numbers in the range of the space[7]. The new inserted subtree could not be treated as Nested Tree if there is no space at the position for the data insertion. On the other hand, it can be treated as Nested Tree if the size of the inserted subtree is larger than the size of the space. Nevertheless, the scope of the new Nested Tree can be extended such as Nested Tree that include the subtree rooted by the parent of the inserted subtree. Figure 4 shows how the subtree is inserted between the node (39,39) and node (41,41). The root of the newly inserted subtree will be labeled with position in between 39 and 41, which will be 40. As such, the label for the subtree will be (40:1, 40:5), and subsequently (40:2, 40:2) for its child.

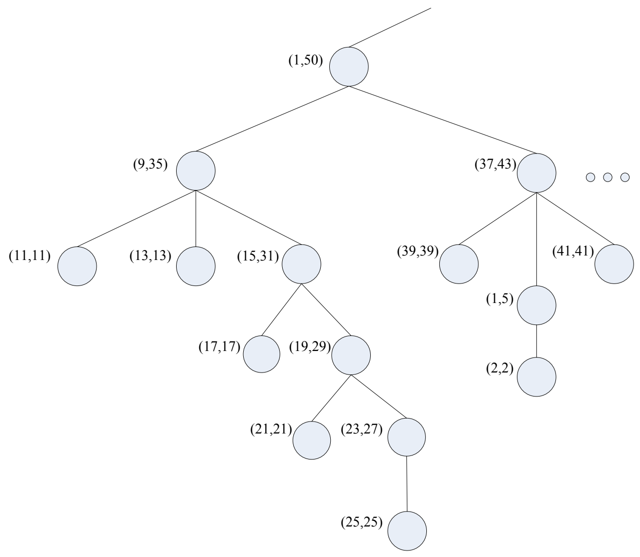


Figure 4. Inserted node labeled with Extended Inverted List.

2.2 ReLab (Region encoding)

ReLab[8] is an example of region encoding. Region encoding uses less resource to label the nodes. Basically, the labeling structure consist of [level, ordinal, rID], where by level represent the number of edges between root node to current node, ordinal is the unique ID that is assigned using pre-order traversal, and rID is the ordinal rightmost sibling. An illustration of ReLab labeling is shown in Figure 5.

Structural relationships are determined using the region property of labeling scheme. For instance, A-D relationship between [1,2,10] and [3,7,10] can be proven by if descendant's rID is within the range of ancestor's ordinal and rID.

Case 1: A-D relationship

"For two nodes n1 and n2 in XML tree, T, n1 is the ancestor of n2 in T if and only if n1(ordinal) < n2(rID) ≤ n1(rID)".

To prove P-C relationship between [1,2,10] and [3,7,10], the difference between the level of child node and level of parent node must be one, and the rID of child must be less or equal to rID of the parent node. As a result, the following lemma is generated.

Case 2: P-C relationship

"For two nodes n1 and n2 in XML tree, T, n1 is the parent of n2 in T if and only if (n1(level)+1 = n2(level) && (n2(rID) ≤ n1rID))".

As for the support for dynamic update, the information could not be found in the current paper [8].

2.3 Order based Scheme (Prefix-based)

OrderBased[9] labeling scheme is based on combination of alphabetical letters and integers describing the level, node order, and parent node order. The label is in the format of duplet with the level and order concatenated as <level order, parentorder>, being (i) level is the distance of any node relatively to the root, (ii) order is specified in alphabetically, order of the node relative to the leftmost node, and finally (iii) parentorder is the order of the parent.

Figure 6 shows the XML tree annotated with OrderBased scheme. The level of the root is 0, the level of the direct child of root is 1, and subsequently, increasing by 1 until the end of hierarchy where the last leaf node is found. As for the order, it is computed using the breath-first traversal with letter 'b', followed by 'c', 'd', subsequently until the 25th and 26th nodes (if present) will be 'z' and 'zb' respectively. The parentorder however, is assigned based on tracing the order of the parent node.

Structural relationship can be determined as follows. For example, in Figure 6, the node with label <1c, a> is the parent of the nodes with labels <2e, c> and <2f, c>. This parent to child relationship is provided because the parent order of the three nodes is "c. As for the sibling, any two nodes that have the same level information and with the same parent order are siblings. For instance, <2e, c>

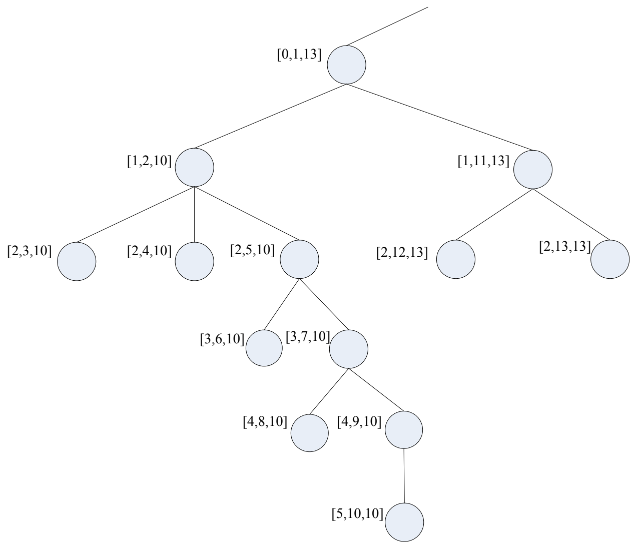


Figure 5. Relab scheme.

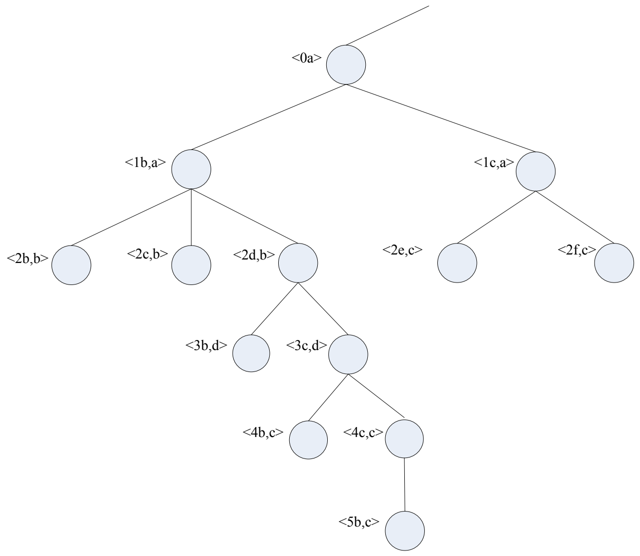


Figure 6. OrderBased labeling scheme.

and <2f, c> are siblings as they are within level 2 and have the same parent order, which is 'c'.

However, to find the ancestors /descendants of a given node, first there is a need to move to the parent/children, and then the parent of the parent/children recursively till the intended level is reached. For example, to find whether node1 <5b, c> have ancestor-descendants relationship with node2 <3c, d>. Firstly, we obtain the parent order of node1, which in this case is 'c'. Then, we find the node in a level higher (in this case, level 4), which have order 'c'. As such, node <4c, c> is retrieved. The process repeated to obtain the parent order of node <4c, c>, which in this case is 'c'. Next, we search the node at a level higher (in this case, level 3), which have order 'c'. As such, node <3c, d> is retrieved. Therefore, node1 <5b, c> and node2 <3c, d> is of ancestor-descendant relationship.

This labeling scheme also supports the dynamic update. For any newly inserted nodes, it works as follows.

1. "To insert a node before the first node of a given level, get the order of the node then count down to the preceding alphabet, if all characters are "b", insert "a" before the last "b". (see Figure 7(a))."
2. "To insert a node between two nodes, keep counting from the code standing before it so that the code for the new node will be greater than the code of its previous sibling and less than the code of its next sibling (see Figure 7(b))."
3. "To insert a node after the last node of a level, increment the order of the last order alphabetically (see Figure 7(c))."

In addition,[9] also proposed optimization routine, named Determine-size to minimize the label size for every level. Using this routine, the optimal characters needed to label the nodes at every level will be computed first. They proved that using optimized method reduces the storage requirement (see Table 1).

2.4 ME Labeling (Multiplicative)

Multiplicative labeling uses odd numbers and multiplication operation to annotate the XML tree. The labeling structure consist of (level, [selflabel, ordinal]), where by level represent the node that is located in the tree, selflabel is the value where parent * ordinal, parent is the selfLabel of parent node, and ordinal is the unique number of the current node[10]. The root node will always be labelled as 1. 2n+1 are used to generate odd numbers for ordinal where n represent the position of a node in the level. For instance, the first node of ordinal at level 1 is 2(1)+1 equal to 3, the second node of ordinal at level 1 is 2(2)+1 equal to 5, the third node of ordinal at level 1 is 2(3)+1 equal to 7 (see Figure 8).

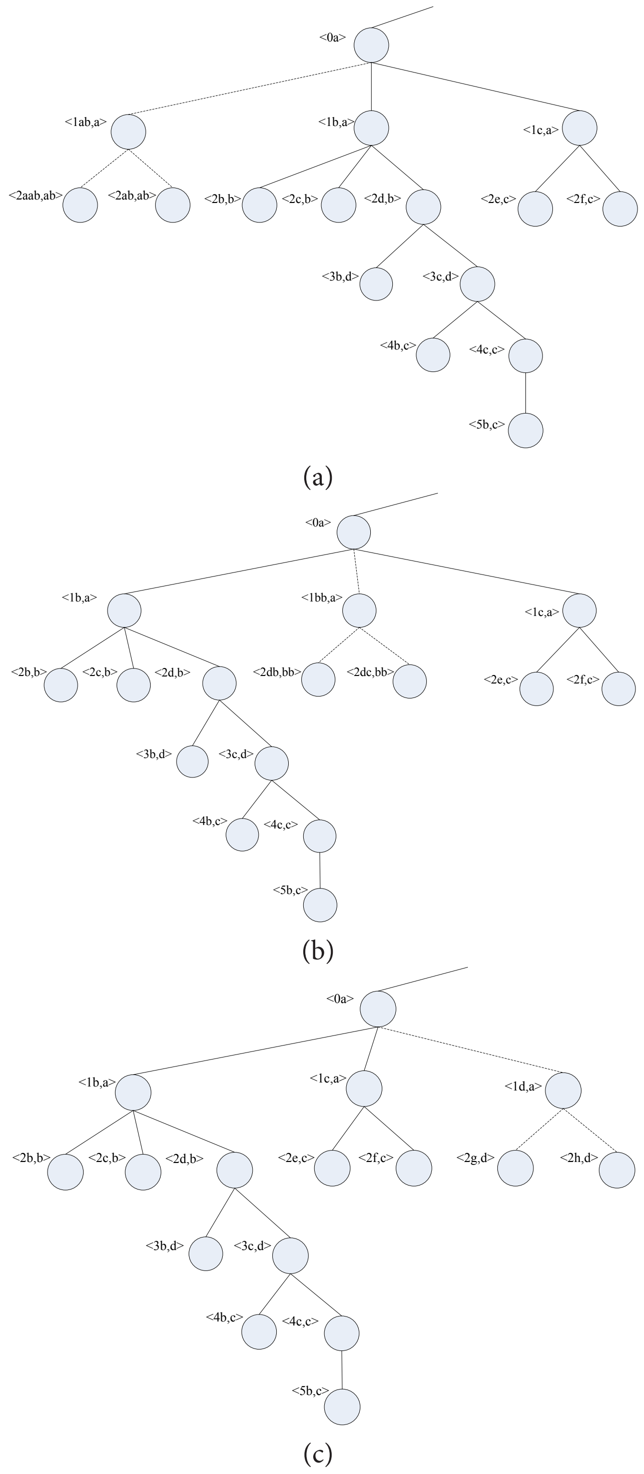


Figure 7. Insertion of subtree in **(a)** the leftmost part, **(b)** between any nodes, **(c)** the rightmost part.

Table 1. Analytical storage requirement[9]

M	Optimized	Un- optimized
24	24	24
50	100	75
75	150	75
100	200	250
1000	3000	20500
2000	6000	81000
1000000	5000000	20000500000

Parent-child relationship (P-C) is the parent label that is inherited to the child label. It can be determine by using the formula selfLabel / ordinal. Parent node is always one level below the child node. For ancestor-descendant relationship (A-D), it can be done using all four condition; condition 1 the node1(selfLabel) must be less than node2(selfLabel), condition 2 the node is divided by node1(selfLabel), condition 3 the Parent(selfLabel) of node2 is divided with node1(selfLabel), and condition 4 the sibling of node2(selfLabel) / node2(selfLabel). When all condition are satisfied, then only A-D relationship is proved that it is valid.

1. *Condition 1: node1(selfLabel) < node2(selfLabel).*
   - Proof: The selfLabel of ancestor is lesser than the selfLabel of descendant: 3 < 525
2. *Condition 2: node2(selfLabel) / node1(selfLabel).*
   - Proof: The selfLabel of descendant , 525 is dividable by the selfLabel of ancestor, 3 (525/3) and the remainder is a 0.
3. *Condition 3: Parent(selfLabel) of node2 / node1(selfLabel).*
   - Proof: The selfLabel of the parent node of descendant (525/3) is 105 is dividable by the selfLabel of ancestor, 3 (105/3), and the remainder is 0.
4. *Condition 4: Sibling of node2(selfLabel) / node2(selfLabel).*
   - Proof: The selfLabel of the sibling node of the parent node, 63 is dividable by the selfLabel of ancestor, 3 and the remainder is 0.

Sibling relationship can be proved by determine the relationship between n1= [3,63,3] and n2 = [3,105,5], first determine the parent of n1 is 63/3 = 21 and parent of n2 is 105/5 = 21. Therefore, sibling relationship is proved between these nodes. On the other hand, there is no additional calculation to be determine for level relationship as it encode in the label itself. For example, in order to identify the level of node [2,15,5], we can simply derive the information from the label which is 2 in this case.

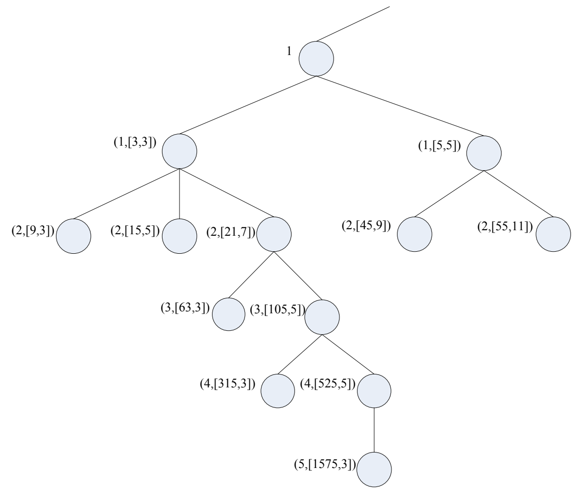


Figure 8. ME Labeling.

Insertion of new node in ME labeling can be assume that a new node is inserted in between NodeA and NodeB, where the selfLabel of NodeA is indicated as selfA and ordinal of NodeA is indicated as ordinalA. Also, selfLabel of NodeB is referred as selfB and ordinal of NodeB is indicated as ordinalB. Suppose that NodeC is the newly inserted node with selfLabel newselfC and ordinal as newordinalC. The generation of newselfC and newordinalC for the NodeC is as follows:

a) newselfC = (selfB)(ordinalA) + (selfA)(ordinalB)
b) newordinalC = newselfC /parent of NodeA or NodeB

For instance, if Node C is inserted in between of Node A (2,[45,9]) and Node B (2,[55,11]) based on Figure 2-13, the generation of new label for Node C is shown as below:

a) newselfC = (55)(9) + (45)(11)
   = 495 + 45
   = 990
b) newordinalC = 990 / 5
   = 198

Thus, the new label for Node C is (2,[990,198]). Figure 9 shows the new inserted node of ME labeling.

As a result, ME labelling does not require relabeling during dynamic updates. The structural relationship between the nodes is maintain by ME labelling even the dynamic update occurred.

2.5 Dynamic XDAS (Hybrid)

Dynamic XDAS[11] is an example of hybrid labeling, which uses binary digits to represent node labels. XDAS generates labels based on the masking technique as shown in

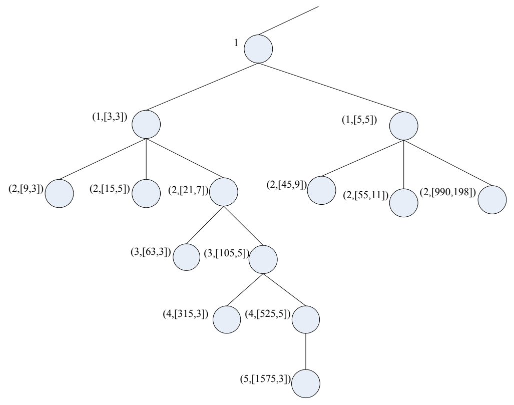


Figure 9. Insertion of new node on ME labeling.

Figure 10. This labeling scheme has two parts: the first part is the level, and the second part is the unique ID generated using bits-masking.

The modified approach from improve binary string labeling (IBSL)[12] into XDAS labels has three main cases as follows.

Case 1: Insert a node before the leftmost node

"In this case, the label of the inserted node is that the label of leftmost node concatenated with the delimiter concatenated with 01."

For instance, from Figure 10, the leftmost has the label 2,01001, so the label of the newly inserted node is going to be 2,01001.01.

Case 2: Insert a node after the rightmost node

"In this case, the label of the inserted node is that the label of rightmost node concatenated with the delimiter concatenated with 11."

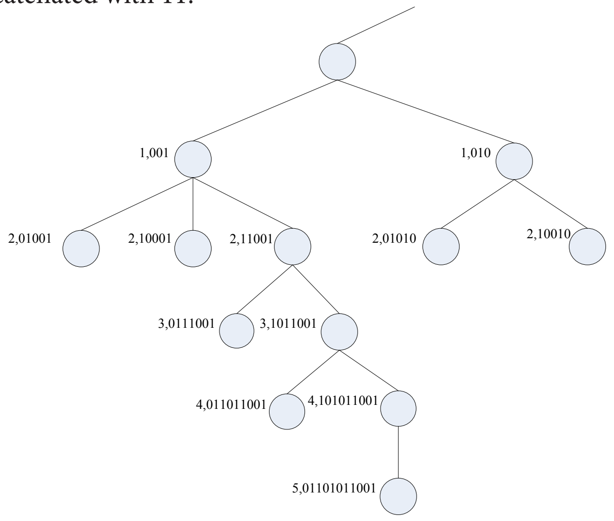


Figure 10. XDAS labeling scheme.

For instance, from Figure 10, the rightmost has the label 2,10010, so the label of the newly inserted node is going to be 2,10010.11.

Case 3: Insert a node between any two nodes at any position

"In this case, the label of the inserted node depends on the size of the labels of the two neighbour sibling nodes." Figure 11 shows the different cases could happen whenever a new node to be inserted at any position, as follows:

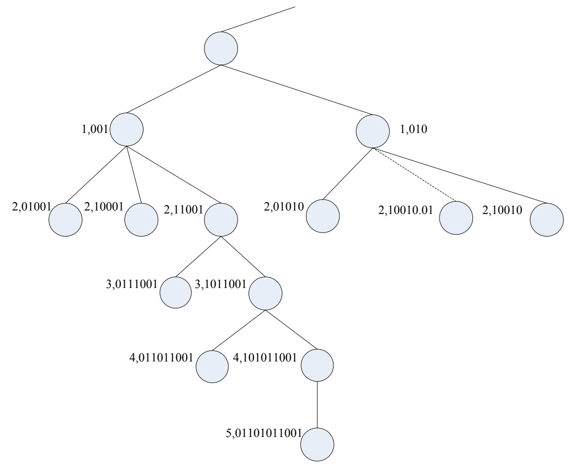
a) "If the size of the left sibling node's label is less than or equal to the size of the right sibling node's label, then the label of the inserted node is the label of the right sibling node concatenated with the delimiter concatenated with 01."

   For instance, from Figure 10, the left sibling node has the label 2,01010 and the right sibling node has the label 2,10010, so the label of the newly inserted node is going to be 2,10010.01 (see Figure 11 (a)).

b) "If the size of the label of the left sibling node is larger than the size of the label of the right sibling node, then the label of the inserted node is that the label of left sibling node concatenated with "1"." For instance, from Figure 11(a), the left sibling node has the label 2,10010.01 and the right sibling node has the label 2,10010, so the label of the inserted node is going to be 2,10010.011 (see Figure 11 (b)).

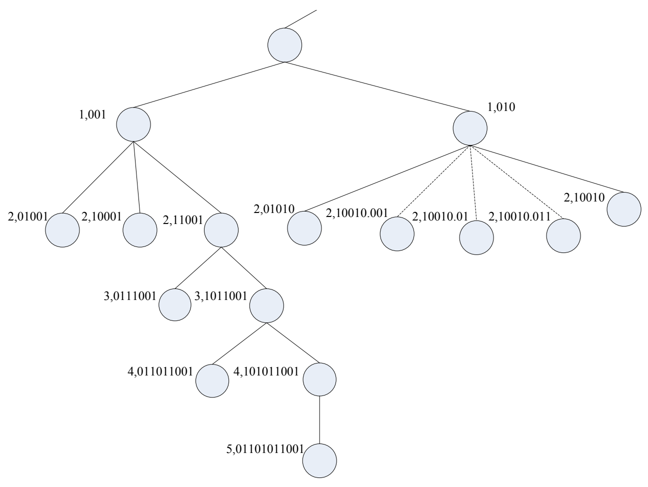
Case 4: Insert a subtree at any position of the tree

"In this case, the root of the inserted subtree is labeled according to the previous cases. The other nodes of the subtree are labeled according XDAS bits-masking technique."



(a)

Figure 11. Insertion node between any two node labels, case **(a)** left sibling node <= right sibling node



(b)

Figure 11. Insertion node between any two node labels, case **(a)** left sibling node <= right sibling node, **(b)** left sibling node > right sibling node.

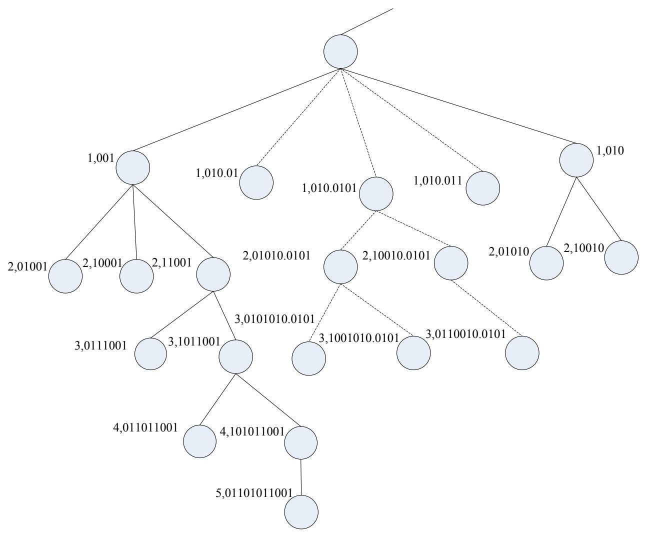


Figure 12 Insertion of subtree at any position of the tree.

Figure 12 shows an example of inserting a subtree into XML tree.

2. Summary and Conclusion

A proper node indexing is essential to enable quick determination of structural relationships such as P-C, A-D and siblings (predecessor and successor) between any two nodes. Choosing a correct indexing is critical. For example, prefix-based indexing scheme size grows too huge as the XML tree goes deeper. The multiplicative labeling suffers from high computation cost. As such, they are not suitable to label a huge XML document. However, they support dynamic updates efficiently for any new possible insertion of subtree/node.

On the other hand, the subtree group has the smallest and usually fixed-length label size. Nevertheless, this scheme is not persistent and robust as large numbers of nodes may need to be relabeled for any update operation.

In recent years, several hybridization of the labeling schemes have been proposed[13-16]. A hybrid scheme integrates the approaches of different schemes with the aim of developing a scheme with the strengths of several schemes.

Table 2 summarizes the advantages and disadvantages of the node indexing mentioned above.

Table 2. Summarization on advantages and disadvantages of node indexing

Labeling scheme	Advantages	Disadvantages
Extended Inverted List[7]	Good for structural join processing.	Existing algorithm uses the same method will not improve the query performance. Reserved space is not enough for the data insertions. Lengths of labels may grow huge.
ReLab[8]	Less complexity in label computations.	Do not support for dynamic update
OrderBased[9]	The label size could be minimized in every level using their proposed routine (Determine-size).	To determine ancestor-descendant, need to recursively trace-back until the intended nodes are compared.
Multiplicative[10]	Structural relationship of XML nodes can be determined easily. Supports dynamic updates without relabeling the nodes.	It will not dynamic update the number of newly inserted nodes that is larger than the reserved numbers.
Dynamic XDAS[11]	Efficient in labeling as it separate node that contain with and without update ID.	Number of bits increase as is goes further down in XML tree.

3. Acknowledgement

This work was partially supported by funding from FRGS, Ministry of Education, Malaysia.

4. References

1. Fang WK. Tree decomposition-based indexing for efficient shortest path and nearest neighbors query answering on graphs. Journal of Computer and System Sciences. 2016; 82(1):23–44.
2. Karthiga D, Gunasekaran S. Optimization of Query Processing in XML Document Using Association and Path Based Indexing. International Journal of Innovative Research in Computer and Communication Engineering. 2013; 1(2):261–66.
3. Cheol-Joo C, Kiseok C, Kwang-Nam C. The XML based Electronic Document Image Retrieval System. Indian Journal of Science and Technology. 2015; 8 (S9):235–9.
4. Alghamdi NS, Rahayu W, Pardede E. Semantic-based Structural and Content indexing for the efficient retrieval of queries over large XML data repositories. Future Generation Computer Systems. 2014; 37:212–31.
5. Hsu WC, Liao IE. CIS-X: A compacted indexing scheme for efficient query evaluation of XML documents. Information Sciences. 2013; 241:195–11.
6. Haw SC, Lee CS. Node Labeling Schemes in XML Query Optimization: A Survey and Open Discussion. IETE Technical Review. 2009; 26(2):89–01.
7. Yun JH, Chung CW. Dynamic interval-based labeling scheme for efficient XML query and update processing. Journal of Systems and Software. 2008; 81(1):56–70.
8. Samini S, Haw SC, Soon LK. ReLab: A Subtree based Labeling Scheme for Efficient XML Query Processing. IEEE International Symposium on Telecommunication Technologies. 2014; 121–25.
9. Assefa BG, Ergenc B. OrderBased Labeling Scheme for Dynamic XML Query Processing. Lecture Notes in Computer Science. 2012; 7465:287–01.
10. Samini S, Haw SC. ME Labeling: A Robust Hybrid Scheme for Dynamic Update in XML Databases. IEEE International Symposium on Telecommunication Technologies. 2014; 126–31.
11. Ghaleb TA, Mohammed S. A Dynamic Labeling Scheme Based on Logical Operators: A Support for Order-Sensitive XML Updates. Procedia Computer Science. 2015; 57:1211–18.
12. Ko HK, Lee S. A binary string approach for updates in dynamic ordered XML data. IEEE Transactions on Knowledge and Data Engineering. 2010; 22(4):602–07.
13. Mirabi M, Ibrahim H, Fathi L. PS+Pre/Post: A novel structure and access mechanism for wireless XML stream supporting twig pattern queries. Pervasive and Mobile Computing. 2015; 15:3–25.
14. Le TN, Ling TW, Jagadish HV, Lu J. Object Semantics for XML Keyword Search. Lecture Notes in Computer Science. 2014; 8422:311–27.
15. Chen Y, Davidson SB, Zheng Y. A bi-labeling based XPath processing system. Information Systems. 2010; 35(2):170–85.
16. Liu J, Ma ZM, Yan L. Efficient labeling scheme for dynamic XML trees. Information Sciences. 2013; 221:338–54.