CHAPTER 3

LITERATURE REVIEW

This chapter presents query processing with XML documents, indexing techniques and current algorithms for generating labels. Here, each labeling algorithm and its limitations are also spotlighted.

3.1 XML QUERY PROCESSING

XML queries are used to select related elements or attributes by specifying the predicate. In a tree-based representation, nodes can represent an element tag, an attribute or a value. Hierarchical relationships like ancestor–descendant and parent-child relationship among XML elements can be represented by edges. Nodes and edges represent conditions. The response of a query must satisfy the conditions specified in the XPath expression. A simple XPath query consists of sequence of nodes in a tree structure. A twig query consists of one or more sub trees in the XPath expression.

XML query processing relies on the conventional top-down or bottom-up tree traversals on the XML document. It is highly inefficient, because it produces large collection of documents. To reduce the overhead of processing queries, indexing or labeling methods are efficiently used. In each method, a specific data structure is used to store the index values and through that the nodes can be accessed. To identify the nodes, each node is assigned with a unique value. These unique values, known as labels, are computed based on the position of the node in the tree structure of the document.
Location Based Schemes

The two broad categories of location based schemes are position based and path based schemes.

In a position based scheme, the location of a node is based on the position of that node. Each node is assigned with a $<\text{preorder, postorder}>$ number. However, a change in the document leads to changes in the values of $<\text{preorder, postorder}>$ of affected nodes. Hence, re-computing the label values is needed.

In Path based scheme or prefix labeling method, (Abiteboul 2001) each node is identified by a label which includes its parent label and self label. Inserting new nodes does not affect the label of the existing nodes. Once the elements and attributes are identified, the query processor can combine the results of various index-based selections to get the desired results.

3.2 INDEXING TECHNIQUES

The general indexing techniques (Meuss 1999), for the XML document are presented in this section.

Classification of Node Labeling Schemes

Node labeling schemes (Haw 2009a) can be broadly classified into four main categories, namely, Subtree labeling, Prefix-based labeling, Multiplicative labeling and Hybrid labeling as shown in Figure 3.1.
Figure 3.1 Labeling Schemes

Subtree Labeling

The label of a node encodes the position and the extent of the subtree that is rooted in that node, by means of offsets in the sequence of nodes resulting from traversing the document tree in a specific order. The relabeling problem occurs, when changes occurs.

The subtree labeling can be divided into two more subclasses: interval encoding and region encoding.

Interval Encoding

Tree Traversal Order :

Each node in XML document is labeled with a pair of unique integers consisting of preorder and postorder traversal sequences. This approach is useful to determine the ancestor-descendant relationship between the nodes easily. It does not find the parent-child relationship. It needs recomputation when changes occur.

Extended Preorder Traversal:

In this scheme, each node is labeled with a pair of numbers <order, size>, where size is an arbitrary value which is greater than the number of current descendant of that node. This scheme is based on the notion of
extended preorder traversal to accommodate the insertions in future. But this size is not enough when skewed insertions take place.

**Region Encoding or Range Encoding:**

In this scheme, each node in XML document is associated with a substring \(<\text{start position, end position}>\) which can be identified by region coordinate (Agarwal 1989), (Li 2001). The start position and end position values are the counts starting from the beginning of the XML document.

**Prefix based Labeling**

The label of a node is encoded with a path from the document root to down to that node. It is interpreted as a sequence to uniquely denote an ancestor of that node on that path. The ancestor-descendant relationships can be found easily from this sequence. It takes more space for encoding and more time to process the long path.

**Multiplicative Labeling**

Atomic numbers are used to identify nodes and the relationships can be computed based on the arithmetic properties of the label values of the nodes. A mapping function maps the unstructured document to a regular tree. Computations are very expensive. Hence, it is unsuitable for large size document.

**Prime Number:**

Prime numbers (Wu 2004) are used to label the XML tree. It uses either bottom-up or top-down approach. In bottom-up approach, the leaf nodes are labeled with prime numbers. The label of a parent node becomes a product of labels of their child nodes. It is not convenient for the nodes having
single child only. In top-down approach, the prime number ‘1’ is assigned for the root node. The label of a node is retrieved from the product of labels of their parent nodes and it’s self-label. A prime number can be used only once. The size of label increases when the document has more levels.

**Hybrid Labeling**

The combination of the various labeling methods may support the fast query processing. Yun et al (2008) developed a scheme called Nested Tree and Nested Inverted List for dynamic interval-based labeling scheme. The sub tree inserted is labeled using traditional interval-based scheme, and other nodes in the sub tree are labeled with a prefix-based numbering scheme. This type of schemes is an efficient one to support a larger class of queries as well as support dynamic updates.

3.3 **EXISTING METHODS**

This section provides an overview of the effect of order-sensitive update and re-computation problem. Also, the current techniques for addressing this problem are also presented.

3.3.1 **Order-sensitive Updates**

The elements in an XML tree are ordered intrinsically which is known as document order. A document order specifies a sequence followed for the elements in that document. The order of elements influences the semantics of XML document. The updating of the nodes (Amagasa 2003), (Fomichev 2006) (Li 2005) (Li 2006a) (O’Neil 2004) causes a problem in keeping the document order in the document. This requires re-computation of node labels. The methods for maintaining the document order in dynamic XML have been advocated by Amagasa et al (2003), Cohen et al (2002), Li
et al (2005), Li et al (2006a), Li et al (2006b) O’Neil et al (2004), Silberstein et al (2005) and Wu et al (2004). The updating problem is shown in the Figure 3.2. The problem is raised because of the semi-structured nature of the XML document. This is explained with the sample document given in Figure 2.1. The node with dotted line indicates that the node is a newly inserted node. The same notation is used for showing the insertion throughout the thesis. If the schema of the sample document encounters an element definition like “course (student)*”, the <course> may have any number of <student> nodes, in any order. Suppose, a new <student> node is inserted before the first child <student> node, then re-computation problem is raised. As the <student> nodes are already having labels as 2 and 5, there is no way of assigning a label to the new node which maintains the document order. Suppose, the label ‘2’ is assigned for the newly inserted node, then the label of all the other nodes in the document should be changed. This is known as re-labeling or re-computation. It is a threat in labeling methods.

![Figure 3.2 Re-computation – insertion before first child](image)

The same problem is raised, if new <student> nodes are inserted between the two consecutive <student> nodes. It is depicted in Figure 3.3. Similarly, if a new node <rollno> is added as a second child of the first <student> node, then labeling the <rollno> becomes a problem. Another problem occurs when the labeling method could not generate enough labels for the newly inserted nodes.
In any order, the `<student>` node can be added as a child of `<course>` node. Hence, the way of adding the nodes in the XML document causes the problem of updating the labels of existing nodes. While inserting such new nodes, the order of siblings is not maintained. But, while querying, result of the query is based on the position of nodes. It needs information about the order of siblings. Hence, it is essential to preserve the sequence of labels.

When the size of the document increases, the size of index also increases. When adding new node between the siblings or before the first child of a node, the re-computation is required. For the large size XML document, it is a big problem. Re-computing the label of XML nodes is a time consuming process. The cost of re-computation includes the number of nodes to be re-labeled and time taken for changing the labels of affected nodes. Suppose, for the frequently changed document, most of the time may be spent only for re-computation process rather than query processing. Various systems proposed different ways to overcome the updating problem and are provided in the following section.
3.3.2 Labeling Methods

Number of query processing approaches such as path indexing, range labeling, tree labeling, numbering scheme etc. has been addressed by Haw et al (2009).

A method Yet Another Path Index (YAPI) developed by Amato et al (2003), supports the evaluation of containment relationships for XML searching. This technique can process the queries with wildcards. It constructs ‘rotated lexicon’, which consists of all possible rotations of all terms in the original lexicon. Suppose, for the term ‘apple’, the rotated lexicon will contain the terms apple^, pple^a, ple^ap, le^app, e^appl, ^apple, where the character ^ denotes the string termination. The rotated lexicon is ordered alphabetically using the sequence ^,a,b,c,…. Using a B+ Tree structure, a search structure is made. A mapping is done between the original lexicon and the rotated versions of all terms in the original lexicon. For instance, the query *pl* is transformed into pl*, which matches the terms pleˆap, pleˆsta, and ployˆem to the original lexicon such as apple, staple and employ. Due to the rotations of all original lexicons, a high storage area is needed. Moreover, this method handles static documents only. Whenever a change occurs in the document, the ordered list may be changed. Hence, it is not suitable for dynamic data.

In range labeling scheme, the label of a node is interpreted as a pair of numbers <start, end> using starting and ending position of a node. From the range values, the ancestor-descendant relationship can be found. A node i with range value <S_i, E_i> is an ancestor of a node j with range value <S_j, E_j> iff S_i<S_j<E_j<E_i. Ancestor-descendants relationship can be determined in constant time. A sample is provided in Figure 3.4.
This method has lack of flexibility. Re-computation problem will occur, when inserting new nodes take place. It is shown in Figure 3.5.

To avoid re-computation, few approaches have been proposed to leave some “gaps” in labels for future use. The gaps are also not enough, when skewed insertions take place.

The Simple Prefix (SP) numbering scheme (Cohen 2002) assigns a specific binary string to each child of an XML document. It includes two approaches. The first approach is one-bit growth. An example is given in Figure 3.6. Let a node v have number of children. The label of the node v is
denoted as \( L(v) \). Then, the first child’s code is 0, and it is labeled as \( L(v) \cdot 0 \). The second child’s code is 10 and has the label \( L(v) \cdot 10 \). The third child’s code is 110 and the label is \( L(v) \cdot 110 \). Thus, the \( i \)-th child’s code is denoted with 1 for each child’s code, together with a 0 at the end.

For any new node \( u \), the label \( L(u) = L(v) + 1 \ldots 10 \)

where \( v \) represents the parent of \( u \). The number of labeled children of \( u \) is \( i \). Root node is labeled as an empty string. The children nodes are assigned with the code values 0, 10, 110 etc.

![Figure 3.6 Simple prefix algorithm-one bit growth](image)

**Figure 3.6 Simple prefix algorithm-one bit growth**

If new nodes \(<\text{student}>\) and \(<\text{rollno}>\) are added in the document, it raises the re-computation problem, which is shown in Figure 3.7.

![Figure 3.7 Re-computation – SP (one bit growth)](image)
In the second approach, double-bit growth, the label is generated as follows. For example, the node \( u_i \)’s code is \( L(v) \cdot L(u_i) \) where \( L(v) \) is its direct parent code. The node \( v \) has children \( u_1, u_2, u_3, \ldots \). It assigns its children as \( L(u_1) = 0 \), \( L(u_2) = 10 \), \( L(u_3) = 1100 \), \( L(u_4) = 1101 \), \( L(u_5) = 1110 \), \( L(u_6) = 11110000 \), etc. Generally, the code \( L(u_{i+1}) \) of the node \( u_{i+1} \) is derived by increasing \( L(u_i) \) value by 1. The code value of the children nodes are assigned with the values 0, 10, 1100, 1101, 1110, 11110000, etc. Whenever the label having all ones, then it doubles the length of the label by adding a sequence of zeros. It does not need renumbering for any insertion sequence. In both approaches, the index size value grows very rapidly. In the worst case, the space complexity is \( O(N^2) \), where \( N \) is the number of nodes in an XML tree.

Dewey prefix-based numbering scheme provides (Tatarinov 2002) a technique in which the label value of a child is based on the label value of its parent node and the position of the child node. The technique is presented as follows:

The code value of a node \( v \) is coded as \( L(v) \). Let ‘\( n \)’ children of a node \( v \), are \( u_1, u_2, u_3, \ldots u_n \). Then, the code of the child node \( u_i \) is \( L(u_i) = L(v) \cdot i \). In Figure 3.8, first child \(<\text{student}>\) of root node \(<\text{course}>\) is coded as 1.1. The second child of that node \(<\text{student}>\) is \(<\text{marks}>\). Then, the code value of that node \(<\text{marks}>\) is 1.1.2. Suppose, two new nodes \(<\text{student}>\) and \(<\text{rollno}>\) are inserted in the document as shown in Figure 3.8, the prefix numbering scheme fails to assign labels for these nodes. It needs recomputation of labels.
Figure 3.8 Re-computation in Dewey prefix method

GRoup based Prefix (GRP) developed by Jiaheng et al (2004) associates each node $n$ in the XML tree with a pair of number $<$groupID, prefix-label$, where groupID is a non-negative integer and prefix-label is a binary string. Here, group represents a set of sub trees having the common parent node. All nodes in the same group have the same groupID and are distinguished by their prefix-labels. Given an XML tree $T$, a group is a set of sub trees. All root nodes of sub trees in this set have the common parent node in $T$. An example is shown in Figure 3.9. Suppose the groupID of nodes C, D, E are 2, then these have the label values “2.0”, “2.10”, “2.110”. As can be seen in the Figure 3.9, the dynamic changes in the document need re-computation of the existing labels.

Figure 3.9 Groups in GRP labeling scheme
Hye-Kyeong Ko et al (2010) suggested the method name called Extended Improved Binary String Labeling. In this method, the label values are compared based on their lexicographical order instead of numerical order.

Whenever changes occur in the document, the lexicographic order of the consecutive labels is being kept and it avoids re-computation. However, it cannot fully avoid the increase in label size of the inserted nodes in case of continuous insertions. The label size of the inserted node increases rapidly.

Another scheme, Prefix-based PBi Tree (Wang 2003a) addresses the updating problem in dynamic XML. In this scheme, the unique code of a child node is determined from its parent code and the number of bits in the code is less than the number of child nodes. It uses preserved codes between every two child nodes to reduce the possibility of renumbering.

Suppose a node v has ‘n’ child nodes, u₁, u₂, u₃ ... uₙ. A unique child code L’(uᵢ) is assigned to uᵢ using m-bits such that 2m-1 < n ≤ 2m. Let the parent node v’s code be L(v), then the code of uᵢ is L(uᵢ) = L(v)-L’(uᵢ) where “-” is concatenation operator. A sample document is shown in the Figure 3.10.

Figure 3.10 PBi tree labeling scheme
This scheme does not completely avoid the possibility of re-computing labels for all siblings and their descendants, when changes occurred. This is depicted in the Figure 3.11. The nodes ‘student’ and ‘rollno’ do not have the label values. For assigning labels to these nodes, there is no enough number for generating label. Similarly, Wang et al (2003b) developed another scheme called ViST, which is a dynamic index and used for querying XML data.

![Figure 3.11 Re-computation in PBi tree labeling scheme](image)

The numbering scheme proposed by Li et al (2001), assigns a pair of numbers <order, size> for each node. The order of a parent node is less than the order of its child node. The size value of a node can be an arbitrary integer larger than the total number of its current descendants. In this scheme, inserting a node requires re-computation of the affected nodes. The ancestors and descendants of newly inserted nodes in the XML tree structure are affected. It does not totally avoid the possibility of re-computation of labels of the nodes.

Amagasa et al (2003) developed a labeling scheme to avoid re-computation problem by using floating point values with the starting and ending intervals. While frequent update occurs, this scheme cannot avoid the re-computation.

Wu et al (2004) illustrated Prime Number labeling scheme for labeling XML trees. When insertions are made, it needs to re-compute the existing label values.

Also, Mohammad et al (2010a) (2010b) developed various indexing schemes based on the levels of a document. Xu et al (2007) have been used vectors for storing indexes for the dynamic XML. Rao et al (2004) developed a scheme, PRIIX, used for indexing and querying XML data by using Prufer sequences. Weigel et al (2005) suggested one numbering scheme for XML documents, the tree databases are used for deciding and reconstructing relations in Tree.

Li et al (2005) created an encoding scheme using binary strings. Based on this scheme, Li et al (2006a), (2006b), demonstrated that the insertion of nodes in a specific sequence in XML document does not affect the document order. But, the size of label increases by 2 bits for each newly inserted nodes in QED.

Another scheme namely Labeling Scheme for Dynamic XML data (LSDX) (Maggie 2005) uses codes to specify the label of each node. The numbers (0-9) and letters (a-z) are combined in this labeling scheme to label XML trees. The numbers are used to represent the levels whereas the letters (a-z) are used to define label for the nodes. This makes the scheme compact and persistent. The advantages include a) it supports representation of different relationships among the nodes. b) it is helpful for quick insert, delete and update operations.
In LSDX, consider a node $v$ with $n$ child nodes, $u_1, u_2, \ldots, u_n$, a unique code of $u_i$ consists of its level followed by the code of its parent node and ".". The labels of child nodes are in alphabetical order. The method is shown in Figure 3.12.

![Figure 3.12 LSDX labeling method](image)

If the document grows either vertically or horizontally, the size of the labels becomes too large. In Figure 3.13, the growth of the label length is illustrated.

![Figure 3.13 Growth of labels in LSDX](image)

But, in few occasions, this scheme leads to collisions between labels. Consider that the existing <student> nodes at the same level are having the labels ‘1a.b’, ‘1a.z’, ‘1a.zb’, and ‘1a.zc’. A new <student> node is added after second <student> node. Similarly, another new <student> node is
added after third `<student>` node. Then, new nodes are having the same labels ‘1a.zbb’. That is, collisions occur in labeling. LSDX could not generate unique persistent labels. Thus, the re-computation problem is not avoided using this method.

In LSDX, the scaling factors 0.01 to 0.1 are used to generate synthetic XMark datasets (Schmidt 2002) (Martin 2002) of different sizes. Generated documents are named as x1, x2,..., x10. The properties of these documents, Size of the label generated and time taken for generating labels are as shown in Table 4.1, Table 4.3 and Table 4.5 respectively and given in chapter 4.

Although, the above mentioned methods provide query processing, they require high query response time. Some of the methods avoid re-computation of label of the existing nodes, but the size of the labels is large. While querying XML document, handling large size label becomes a big problem. Also, considerable amount of time is needed for generating the labels. It is also essential to preserve the relationships among the nodes.

To address the above issues, this thesis proposes three approaches for labeling nodes of XML document. The labels generated in these methods occupy less space as well as preserve relationships between the nodes. And, these approaches avoid the re-computation of labels of existing nodes. Experimental results shows that an improvement in query processing system.

The next chapter deals with the first approach, New Labeling Scheme for XML (NLSX), in detail.