DDE: From Dewey to a Fully Dynamic XML Labeling Scheme

Liang Xu, Tok Wang Ling, Huayu Wu, Zhifeng Bao
School of Computing
National University of Singapore
{xuliang,lingtw,wuhuayu,baozhife}@comp.nus.edu.sg

ABSTRACT

Labeling schemes lie at the core of query processing for many XML database management systems. Designing labeling schemes for dynamic XML documents is an important problem that has received a lot of research attention. Existing dynamic labeling schemes, however, often sacrifice query performance and introduce additional labeling cost to facilitate arbitrary updates even when the documents actually seldom get updated. Since the line between static and dynamic XML documents is often blurred in practice, we believe it is important to design a labeling scheme that is compact and efficient regardless of whether the documents are frequently updated or not. In this paper, we propose a novel labeling scheme called DDE (for Dynamic DEwey) which is tailored for both static and dynamic XML documents. For static documents, the labels of DDE are the same as those of dewey which yield compact size and high query performance. When updates take place, DDE can completely avoid re-labeling and its label quality is most resilient to the number and order of insertions compared to the existing approaches. In addition, we introduce Compact DDE (CDDE) which is designed to optimize the performance of DDE for insertions. Both DDE and CDDE can be incorporated into existing systems and applications that are based on dewey labeling scheme with minimum efforts. Experiment results demonstrate the benefits of our proposed labeling schemes over the previous approaches.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query processing

General Terms

Algorithms, Performance

Keywords

Dynamic XML, Labeling scheme, Update, Dewey

SIGMOD'09, June 29–July 2, 2009, Providence, Rhode Island, USA.

1. INTRODUCTION

The rise of XML[5] as a de facto standard for data exchange and representation has generated a lot of interest on querying XML documents that conform to an *ordered tree-structured* data model. Labeling schemes facilitate XML query processing by assigning each node in the XML tree a unique label. The structural relationships of the nodes, such as Parent/Child (PC), Ancestor/Descendant (AD) and document order, can be efficiently established by comparing their labels.

Dewey[3][6][13] labeling scheme has been widely adopted in XML query processing. A dewey label of a node represents the path from the document root to the node. In addition to PC, AD and document order, Dewey supports efficient determination of sibling relationships as well. Moreover, due to the path information contained in its labels, dewey has also become the natural choice for XML keyword query processing[16][12] which has a great interest in computing the Lowest Common Ancestor (LCA) for a set of nodes. Containment labeling scheme[17][10], which is also popular in many applications, does not support the determination of sibling relationships or the computation of LCA.

While dewey labeling scheme works well for static XML documents, it may suffer from high cost of re-labeling for dynamic XML documents where nodes can be arbitrarily inserted and deleted. ORDPATH[11], which is used in the latest versions of Microsoft® SQL Server™, is based on dewey and designed for dynamic XML documents. In ORDPATH labeling scheme, only odd numbers are used at initial labeling. Processing insertions with ORDPATH labels is based on a special 'careting in' technique where even numbers are not counted as components that increase the level of a node. While ORDPATH supports insertion of new nodes at arbitrary positions in the XML tree, the flexibility comes with inseparable costs even when the XML documents seldom get updated: a) Compared with dewey, skipping even numbers makes ORDPATH labels less compact; and b) The 'careting in' technique introduces additional complexity for ORDPATH label processing. For example, level information, which can be easily inferred from a dewey label, can not be deduced from an ORDPATH label unless the parities of all its components are checked.

Recently several encoding schemes[8][7][15] have been proposed as a new approach to process updates in XML documents. An encoding scheme transforms the labels from their original format to another format which allows dynamic updates without re-labeling. Compared with the previous works on labeling dynamic XML documents, the en-

coding approach has shown better performance when the XML documents are frequently updated[9].

However, the encoding approach is no magic bullet. First of all, transforming labels into dynamic formats incurs extra labeling cost. In addition, it is common for an XML repository to have XML documents that are frequently updated and those that are not. If an encoding scheme is applied only to those dynamic XML documents, we have to face a situation where different documents may have different label formats. As a result, different storage and query mechanisms need to be enforced, making updating and querying complicated. To make matters worse, the system administrator bears the burden of deciding which of the documents are dynamic or static. This in general is a difficult, if not impossible task as the updating frequency of a document can vary according to time: a document can, for example, be frequently updated for a period of time and remains unchanged after that. To avoid this situation, it may be tempting to apply an encoding scheme to all the documents. However, this solution leads to extra encoding cost and most importantly, it is wasteful that the static documents have to adopt labels in dynamic formats as they are more efficiently supported by the static labeling schemes.

In this paper, we revisit the concept of dewey labeling scheme. By defining a novel ordering concept, we are able to transform dewey into a fully dynamic labeling scheme: Dynamic DEwey (DDE). Compared with the previous labeling schemes, a distinguishing feature of DDE is that it is tailored for both static and dynamic XML documents. In particular, the labels of DDE are the same as those of dewey if no update takes place, yielding compact size and high query performance. Our contribution in this paper can be summarized as follows:

- We propose a novel labeling scheme: Dynamic DEwey (DDE) which can efficiently support queries for both static and dynamic XML documents. DDE can completely avoid re-labeling when updating the XML documents.
- We introduce a variant of DDE, namely Compact DDE (CDDE), to optimize the performance of DDE for insertions.
- Both DDE and CDDE can be very easily incorporated into existing systems and applications that adopt dewey labeling scheme to support efficient update and query processing.
- Extensive experiments are conducted to demonstrate the benefits of our proposed labeling schemes over previous approaches.

The rest of the paper is organized as follows. Section 2 introduces the background and related work. In Section 3, we describe our DDE labeling scheme in details, illustrating how DDE handles updates without re-labeling, followed by the proof of correctness of our algorithm. Section 4 describes a variant of our DDE labeling scheme which produces more compact labels than DDE after insertions. In Section 5, we address the problem of efficiently computing the various relationships based on our proposed labeling schemes. We present our experimental study in Section 6 and conclude the paper in Section 7.

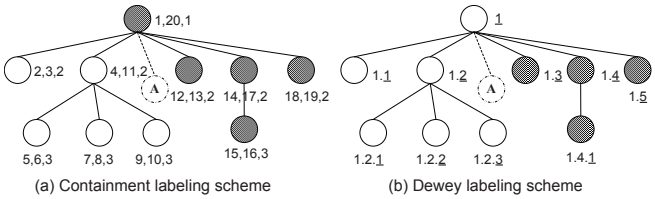


Figure 1: Labeling an XML tree

2. BACKGROUND AND RELATED WORK

In this section, we introduce the existing works on labeling static and dynamic XML documents with a special focus on dewey labeling scheme which is most related to our work.

2.1 Containment labeling scheme

Containment labeling scheme is the representative of range-based labeling schemes. As shown in Figure 1 (a), a containment label consists of three values: *start*, *end* and *level* where *level* indicates the level of its corresponding node in the XML tree. *start* and *end* values of a node define a range which contains the ranges of all its descendant nodes. For example, node 1,20,1 is an ancestor of node 7,8,3 as $1 < 7 < 8 < 20$. Two nodes have PC relationship if, in addition to AD relationship, their level difference is 1. Document order can also be efficiently deduced from two containment labels by comparing their *start* values.

However, containment labeling scheme can not support updates efficiently. As shown in Figure 1 (a), insertion of node A after the second child of the root leads to the re-labeling of all the five shaded nodes. In general, when a node is inserted into an XML tree with containment labels, all its ancestor nodes and all the nodes after this node in document order need to be re-labeled. Leaving gaps at the initial labeling[10] can not solve this problem as re-labeling is still necessary after the gaps are filled. Moreover, gaps make initial labels less compact and therefore increase their storage costs. Floating point numbers have been suggested to be used instead of integers for containment labels to avoid re-labeling[4]. However, the precision of a floating point number is limited because its mantissa is represented by fixed number of bits in a computer. As a result, re-labeling is still unavoidable after the number of insertions exceeds certain limit.

2.2 Dewey labeling scheme

Dewey labeling scheme assigns each node a dewey label which is a concatenation of its parent's label and its local order. As we can see from Figure 1 (b), a dewey label is a sequence of components separated by '.' where the last component of the sequence (the number underlined) represents the local order of the node. The sequence of components before the last component is called the *parent label* of the node as it is inherited from its parent node. The local order of a node is i if it is the i^{th} child of its parent. Unlike containment labels which have *level* fields, the level information is implicitly represented by a dewey label, that is, the number of components in the label. We denote the number of components of a dewey label A as $|A|$.

Dewey labels are ordered by dewey order. Given two dewey labels $A : a_1.a_2 \ldots a_m$ and $B : b_1.b_2 \ldots b_n$, we define dewey order (denoted as $\prec_{dewey}$) as:

DEFINITION 1 (DEWEY ORDER). *$A \prec_{dewey} B$ if and only if one of the following two conditions holds:*

C1. $m < n$ *and* $a_1 = b_1, a_2 = b_2, \ldots, a_m = b_m$.

C2. $\exists k \leq min(m, n)$, *such that* $a_1 = b_1, a_2 = b_2, \ldots, a_{k-1} = b_{k-1}$ *and* $a_k < b_k$.

Let A and B be two distinct dewey labels, we have either $A \prec_{dewey} B$ or $B \prec_{dewey} A$. Note that dewey order can be seen as strict lexicographical order, i.e. $A \prec_{dewey} B$ if and only if A precedes B in lexicographical order and $A \neq B$.

The relationships of A and B can be established based on the following properties:

P1 (AD RELATIONSHIP). A is an ancestor of B if and only if $m < n$ and $a_1 = b_1, a_2 = b_2, \ldots, a_m = b_m$, i.e. $a_1.a_2 \ldots a_m$ is a prefix of $b_1.b_2 \ldots b_n$.

P2 (PC RELATIONSHIP). A is the parent of B if and only if A is an ancestor of B and $m = n - 1$, i.e. $a_1.a_2 \ldots a_m$ matches the parent label of $b_1.b_2 \ldots b_n$.

P3 (DOCUMENT ORDER). A precedes B in document order if and only if $A \prec_{dewey} B$.

In addition, dewey labels support determining sibling relationships as well as computing Lowest Common Ancestor (LCA) which containment labels do not.

P4 (SIBLING RELATIONSHIP). A is a sibling of B is A's parent label matches B's parent label.

P5 (LCA). The LCA of A and B is C, such that C is an ancestor of both A and B, and either (1) $|C| = min(m, n)$, or (2) $a_{(|C|+1)} \neq b_{(|C|+1)}$.

A simple extension of sibling relationship is preceding/ following sibling relationship which we ignore here.

For dewey labeling scheme, insertion of a new node can incur heavy cost of re-labeling as illustrated in Figure 1 (b). All the four shaded nodes have to be re-labeled as a result of inserting node A. In general, when a node is inserted into the XML tree with dewey labels, all its following siblings as well as their descendants need to be re-labeled.

ORDPATH [11] is based on dewey and uses only odd numbers at the initial labeling. Even numbers are reserved for insertions and only used as 'caret's. To insert between two ORDPATH labels whose last components are consecutive odd numbers, the new label is generated using an additional even number which falls between the two odd numbers. We refer to this as the 'careting in' technique. For example, to insert between two ORDPATH labels 1.3 and 1.5, we use 4 which is the even number between 3 and 5 as the 'caret'. The new label is 1.4.1 where 4, the caret, is not counted as a component that increases the level of a node.

Based on the 'careting in' technique, each level in an ORDPATH label is possibly represented by a variable number of even numbers followed by an odd number. This property complicates the processing of ORDPATH labels and therefore negatively affects the query performance. For example, computing the LCA of dewey labels is equivalent to finding the longest common prefix of them. For ORDPATH labels, however, extra care has be to taken to make sure the LCA is a valid ORDPATH label. As an example, the longest common prefix of two ORDPATH labels 1.6.2.1 and 1.6.2.3.5 is 1.6.2 whereas their LCA should be 1. The complexity introduced by the 'careting in' technique *fundamentally* affects the query processing with ORDPATH labels even if no update actually takes place.

2.3 Prime labeling scheme

Prime labeling scheme[14], unlike containment and dewey labeling schemes, is designed to accommodate dynamic insertions without re-labeling. In prime labeling scheme, each node is associated with a unique prime number ($self\_label$). The label of a node is a number which is the product of its $self\_label$ and the label of its parent node ($parent\_label$). Since all $self_label$s are distinct prime numbers, the factorization of a label can be used to identify a unique path in an XML tree. Given two nodes n and m, n is an ancestor of m if and only if $label(m)$ mod $label(n)$=0. n is the parent of m if and only if $label(n) = label(m)/self\_label(m)$.

To determine document order, prime labeling scheme uses an SC (Simultaneous Congruence) value to derive the mapping from $self_label$s to global orders. In practice, to prevent the SC value from getting too large, a list of SC values is used where each SC value maintains the global ordering of five nodes. For large XML documents, the list of SC values can be very long, making its storage and maintenance expensive. Whenever a node is inserted or deleted, on average half of the SC values have to be re-calculated based on Euler's quotient function, which has been shown to be very time consuming[7].

2.4 The Encoding Approach

Several encoding schemes[8][7][15] have been proposed to facilitate efficient updates in XML documents. An encoding scheme is orthogonal to a particular labeling scheme and transforms its labels from the original format to some dynamic format.

QED[7] encoding scheme transforms labels to QED codes. Given the set of numbers $A = \{1, 2, 3\}$ where each number can be stored with 2 bits, a QED code is a sequence of the elements in A that ends with 2 or 3. QED codes are dynamic in the sense that, given any two QED codes, we can always find another QED code which falls between them in lexicographical order. However, the lengths of QED codes increase very fast for skewed insertions. For example, if we keep inserting before a QED code 32, the new QED codes are 312, 3112, 31112..., with 2 bits increase in code length per insertion. The fast increase of code lengths can have a significant negative impact on the storage cost as well as update and query performance of QED. CDBS[8] is similar to QED except that the unit for lexicographical order is one bit, that is, 0 or 1. CDBS is more compact and more efficient to process than QED, but can encounter overflow problem as it uses fixed bits to represent its length. Vector[15] labels are less compact than QED, but scales better for skewed insertions.

The application of an encoding scheme is to transform the order-sensitive components of the original labels (e.g. *start* and *end* in containment labels) to dynamic codes such that the transformation is order-preserving and the resulting label size is as small as possible. However, the encoding processes of existing encoding schemes can be costly for the following reasons: a) They involve the comparison and manipulation of dynamic codes of variable lengths; b) For large XML documents, the encoding tables created by the encod-

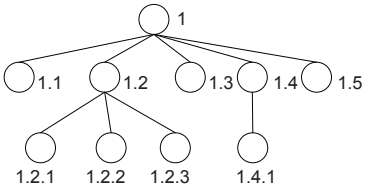


Figure 2: DDE initial labeling

ing process can be of huge sizes; and c) When applying to dewey labeling scheme, the encoding process is further complicated by the fact that every component in a dewey label needs to be encoded, each based on a possibly different encoding table. In summary, the encoding approach suffers from its inherent cost of encoding which makes it less attractive for documents that are not frequently updated.

3. DYNAMIC DEWEY (DDE)

In this section, we describe our DDE labeling scheme in details.

3.1 DDE initial labeling

Every DDE label is a sequence of components that represents a unique path from the document root to a node. More specifically, given a DDE label of the form $a_1.a_2 \ldots a_m$, its parent label and local order are $a_1.a_2 \ldots a_{m-1}$ and a_m respectively. We denote the number of components of a DDE label A as $|A|$.

As illustrated in Figure 2, the initial labeling of our DDE labeling scheme is the same as dewey. The DDE label of the root node is 1, with its parent label being empty. Assume a node in the XML tree has DDE label $a_1.a_2 \ldots a_m$, then the DDE label of its i^{th} child is $a_1.a_2 \ldots a_m.i$. We postpone the proof of correctness of the initial labeling to Section 3.3.1.

3.2 DDE label ordering

DDE label ordering takes into consideration that the first component of a DDE label is restricted to be a *positive* number. This property obviously holds for the initial labels because all of them have 1 as their first components. Moreover, it remains to be valid after random insertions, which we will see in Section 3.5.

First we define preorder (denoted as $A \preceq_{dde} B$) on DDE labels as:

DEFINITION 2 (PREORDER). *Given two DDE labels $A : a_1.a_2 \ldots a_m$ and $B : b_1.b_2 \ldots b_n$, $A \preceq_{dde} B$ if and only if one of the following two conditions holds:*

C1. $m \leq n$ and $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$

C2. $\exists k \leq min(m,n)$, such that $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_{k-1}}{b_{k-1}}$ and $a_k \times b_1 < b_k \times a_1$.

To verify $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$, we compare $\frac{a_1}{b_1} = \frac{a_2}{b_2}$, $\frac{a_1}{b_1} = \frac{a_3}{b_3}$,..., and $\frac{a_1}{b_1} = \frac{a_m}{b_m}$. To take division by 0 into account, we define $\frac{a_1}{b_1} = \frac{a_2}{b_2}$ to be true if $a_1 \times b_2 = a_2 \times b1$ (note that $a_1, b_1 > 0$). For example, $\frac{1}{2} = \frac{0}{0}$, but $\frac{1}{2} \neq \frac{1}{0}$.

Preorder is both reflexive and transitive, as formalized in the following two theorems.

THEOREM 3.1 (REFLEXIVITY OF PREORDER). *Let $A : a_1.a_2 \ldots a_m$ be a DDE label, then $A \preceq_{dde} A$.*

PROOF. $A \preceq_{dde} A$ because C1 in Definition 2 is satisfied, i.e. $m \leq m$ and $\frac{a_1}{a_1} = \frac{a_2}{a_2} = \ldots = \frac{a_m}{a_m} = 1$. □

THEOREM 3.2 (TRANSITIVITY OF PREORDER). *Given three DDE labels $A : a_1.a_2 \ldots a_l$, $B : b_1.b_2 \ldots b_m$ and $C : c_1.c_2 \ldots c_n$, such that $A \preceq_{dde} B$ and $B \preceq_{dde} C$, then $A \preceq_{dde} C$.*

PROOF. From Definition 2, we see that $\preceq_{dde}$ can imply one of two conditions. We consider the case where $A \preceq_{dde} B$ and $B \preceq_{dde} C$ both imply C2. The rest of the cases are simpler and ignored here.

Since $A \preceq_{dde} B$, we have $\exists j \leq min(l,m)$, such that $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_{j-1}}{b_{j-1}}$ and $a_j \times b_1 < b_j \times a_1$. $B \preceq_{dde} C$ implies $\exists k \leq min(m,n)$, such that $\frac{b_1}{c_1} = \frac{b_2}{c_2} = \ldots = \frac{b_{k-1}}{c_{k-1}}$ and $b_k \times c_1 < c_k \times b_1$. Moreover, we assume that $\frac{a_1}{b_1} = \beta$ and $\frac{b_1}{c_1} = \gamma$. We consider the following cases:

j<k. Given $\frac{a_1}{c_1} = \frac{a_1}{b_1} \times \frac{b_1}{c_1} = \beta \times \gamma$, it follows that $\frac{a_1}{c_1} = \frac{a_2}{c_2} = \ldots = \frac{a_{j-1}}{c_{j-1}} = \beta \times \gamma$. From $a_j \times b_1 < b_j \times a_1$ and $\frac{b_1}{c_1} = \frac{b_j}{c_j}$, we have $a_j \times c_1 < \frac{b_j \times a_1 \times c_1}{b_1} = \frac{a_1 \times b_1 \times c_j}{b_1} = c_j \times a_1$. Thus, $A \preceq_{dde} C$.

j=k. Given $\frac{a_1}{c_1} = \frac{a_1}{b_1} \times \frac{b_1}{c_1} = \beta \times \gamma$, it follows that $\frac{a_1}{c_1} = \frac{a_2}{c_2} = \ldots = \frac{a_{j-1}}{c_{j-1}} = \beta \times \gamma$. Then $a_j \times b_1 < b_j \times a_1$ and $b_j \times c_1 < c_j \times b_1$ together imply $a_j \times c_1 < \frac{b_j \times a_1 \times c_1}{b_1} < \frac{c_j \times b_1 \times a_1}{b_1} = c_j \times a_1$. Thus, $A \preceq_{dde} C$.

j>k. From $\frac{a_1}{c_1} = \frac{a_1}{b_1} \times \frac{b_1}{c_1} = \beta \times \gamma$, we have $\frac{a_1}{c_1} = \frac{a_2}{c_2} = \ldots = \frac{a_{k-1}}{c_{k-1}} = \beta \times \gamma$. $\frac{a_1}{b_1} = \frac{a_k}{b_k}$ and $b_k \times c_1 < c_k \times b_1$ together imply that $a_k \times c_1 = \frac{b_k \times a_1 \times c_1}{b_1} < \frac{c_k \times b_1 \times a_1}{b_1} = c_k \times a_1$ and therefore $A \preceq_{dde} C$.

In all the three cases, we have $A \preceq_{dde} C$. □

Based on preorder, we define *equivalence relation* as:

DEFINITION 3 (EQUIVALENCE RELATION). *Two DDE labels A and B have equivalence relation (denoted as $A =_e B$) if and only if $A \preceq_{dde} B$ and $B \preceq_{dde} A$.*

LEMMA 3.1. *Given two DDE labels $A : a_1.a_2 \ldots a_m$ and $B : b_1.b_2 \ldots b_n$, $A =_e B$ if and only if $m = n$ and $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$.*

PROOF. From Definition 3, $A =_e B$ implies that $A \preceq_{dde} B$ and $B \preceq_{dde} A$, which can only be true at the same time if they both satisfy C1 in Definition 2. That is to say, $m \leq n$ and $n \leq m$ should hold. Consequently, we have $m = n$ and $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$. □

DEFINITION 4 (INEQUIVALENT SET). *We say that a set of DDE labels is inequivalent if there does not exist two DDE labels in the set with equivalence relation.*

Finally we are ready to define *DDE order* (denoted as $A \prec_{dde} B$):

DEFINITION 5 (DDE ORDER). *Given two DDE labels A and B, $A \prec_{dde} B$ if and only if $A \preceq_{dde} B$ and $A \neq_e B$.*

The next lemma directly follows from Definition 5.

LEMMA 3.2. *Given two DDE labels $A : a_1.a_2 \ldots a_m$ and $B : b_1.b_2 \ldots b_n$, $A \prec_{dde} B$ if and only if one of the following two conditions holds:*

C1. $m < n$ and $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$.

C2. $\exists k \leq min(m,n)$, *such that* $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_{k-1}}{b_{k-1}}$ *and* $a_k \times b_1 < b_k \times a_1$.

It is easy to see that the DDE order is equivalent to dewey order when $a_1 = b_1 > 0$. Let A and B be two distinct DDE labels from an inequivalent set of DDE labels, we have either $A \prec_{dde} B$ or $B \prec_{dde} A$ (not both).

3.3 DDE label properties

Same as dewey, a DDE label implicitly stores the level information as the number of components in that label. This property will remain true after random insertions and deletions.

Given two DDE labels $A : a_1.a_2 \ldots a_m$ and $B : b_1.b_2 \ldots b_n$, we summarize the properties of DDE labels as follows:

P1 (AD RELATIONSHIP). A is an ancestor of B if and only if $m < n$ and $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$. (AD test for the case where $m = 1 < n$ is by default true. As $m = 1$ implies that A is the document root which is the ancestor any other node.)

P2 (PC RELATIONSHIP). A is the parent of B if and only if A is an ancestor of B and $m = n - 1$.

P3 (DOCUMENT ORDER). A precedes B in document order if and only if $A \prec_{dde} B$.

P4 (SIBLING RELATIONSHIP). A is a sibling of B if and only if $m = n$ and $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_{m-1}}{b_{m-1}}$.

P5 (LCA). The LCA of A and B is C, such that C is an ancestor of both A and B, and either (1) $|C| = min(m,n)$, or (2) $a_{(|C|+1)} \times b_1 \neq b_{(|C|+1)} \times a_1$.

Considering the fact that all dewey labels have 1 as their first components, the five properties of DDE labels can be seen as generalizations of the corresponding properties of dewey labels. In particular, DDE labels can be compared exactly like dewey labels if $a_1 = b_1 > 0$, implying that all the initial DDE labels can be treated as dewey labels. In summary, same as dewey labeling scheme, our DDE labeling scheme is tailored for static XML documents because it does not introduce any additional storage cost or processing complexity.

3.3.1 Correctness of initial labeling

LEMMA 3.3. *Based on DDE labeling scheme, the set of initial DDE labels is inequivalent.*

PROOF. We establish the proof by contradiction. Suppose the set of initial DDE labels is not inequivalent, then by Definition 4, there exist two DDE labels $A : a_1.a_2 \ldots a_m$ and $B : b_1.b_2 \ldots b_m$, such that $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m}$. However, since all the initial DDE labels start with 1, it follows that $a_1 = b_1 = 1$ and therefore, $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_m}{b_m} = \frac{1}{1} = 1$. That is, $a_1 = b_1, a_2 = b_2 \ldots a_m = b_m$, which means A and B are the same. We have a contradiction here because all DDE labels are different in initial labeling. □

Since the set of initial DDE labels is inequivalent, it follows that any two of them are comparable with respect to DDE order. In addition, DDE order is equivalent to dewey order for the initial DDE labels because all of them start with 1. The fact that our initial label assignment is the same as dewey implies that document order is correct with respect to dewey order and therefore DDE order. The same reasoning applies to all the other properties of DDE labels.

3.4 DDE label addition

To process dynamic insertions between DDE labels while preserving their relative order, we introduce addition operation on DDE labels. The addition operation is defined on DDE labels with the same number of components.

DEFINITION 6 (DDE LABEL ADDITION). *Given two DDE labels with the same number of components* $A : a_1.a_2 \ldots a_m$ *and* $B : b_1.b_2 \ldots b_m$, $A + B$ *is defined as:*

$$A + B = (a_1 + b_1).(a_2 + b_2) \ldots (a_m + b_m) \quad (1)$$

The following theorem shows an important property of the addition operation.

THEOREM 3.3. *Given two DDE labels* $A : a_1.a_2 \ldots a_m$ *and* $B : b_1.b_2 \ldots b_m$ *such that A is a sibling of B and* $A \prec_{dde} B$, *then* $A \prec_{dde} (A + B) \prec_{dde} B$

PROOF. Since A and B are siblings, we have $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \ldots = \frac{a_{m-1}}{b_{m-1}}$. Therefore $A \prec_{dde} B$ implies $a_m \times b_1 < b_m \times a_1$. Assume $\frac{a_1}{b_1} = \beta$ and equivalently, $a_1 = \beta \times b_1$.

First we prove $A \prec_{dde} (A+B)$. Given $\frac{a_1}{a_1+b_1} = \frac{\beta \times b_1}{\beta \times b_1 + b_1} = \frac{\beta}{\beta+1}$, we have $\frac{a_1}{a_1+b_1} = \frac{a_2}{a_2+b_2} = \ldots = \frac{a_{m-1}}{a_{m-1}+b_{m-1}} = \frac{\beta}{\beta+1}$. In addition, $a_m \times (a_1 + b_1) = a_m \times a_1 + a_m \times b_1 < a_m \times a_1 + b_m \times a_1 = (a_m + b_m) \times a_1$. Thus, $A \prec_{dde} (A + B)$.

The proof of $(A + B) \prec_{dde} B$ is similar so we ignore it here. □

We use the following example to illustrate the properties of DDE labels that have been introduced so far.

EXAMPLE 3.1. *Consider the XML tree in Figure 3, the dotted circles represent the new nodes inserted into the XML tree. Each new node is associated with a letter. The order in which these new nodes are inserted follows from the alphabetical order of their letters. We ignore for now how their labels are generated. Node 1.2 is an ancestor of node I as* $\frac{1}{3} = \frac{2}{6}$ *and* $|1.2| < |I|$. *F is the parent of I as* $\frac{3}{3} = \frac{6}{6} = \frac{5}{5}$ *and* $|F| = |I| - 1$. *H* $\prec_{dde}$ *E as* $\frac{1}{2} = \frac{2}{4}$ *and* $1 \times 2 < 3 \times 1$, *so H precedes E in document order. E is a sibling of F because* $|E| = |F|$ *and* $\frac{2}{3} = \frac{4}{6}$. *In addition, E* $\prec_{dde}$ *F as* $\frac{2}{3} = \frac{4}{6}$ *and* $3 \times 3 < 2 \times 5$. *Note that G=E+F as* $5.10.8 = 2.4.3 + 3.6.5$, *since E is a sibling of F and E* $\prec_{dde}$ *F, we have E* $\prec_{dde}$ *G* $\prec_{dde}$ *F based on Theorem 3.3. To verify, E* $\prec_{dde}$ *G as* $\frac{2}{5} = \frac{4}{10}$ *and* $3 \times 5 < 2 \times 8$, *G* $\prec_{dde}$ *F as* $\frac{5}{3} = \frac{10}{6}$ *and* $8 \times 3 < 5 \times 5$.

3.5 Processing updates

Similar to dewey labels, it is clear that the deletion of DDE labels does not affect the order of the other labels. The challenging part is how to handle insertions without relabeling. Note that, like ORDPATH, we extend the domain of component values of DDE labels to positive number, negative number and 0. However, since ORDPATH only uses odd numbers at initial labeling, its labels are not as compact as DDE and dewey.

First we introduce how DDE labeling scheme processes insertions with an example.

EXAMPLE 3.2. *In Figure 3, node A is inserted before the first child of the root, we get its label 1.0 by decreasing the local order of 1.1 by 1. Node B is then inserted before A and its label is therefore 1.-1. Node C is inserted after the node*

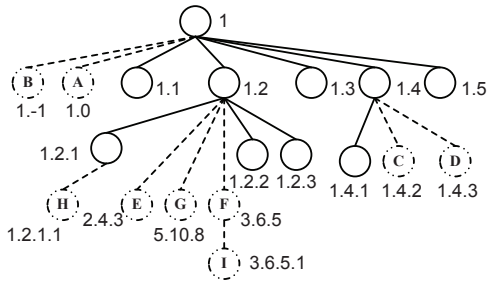


Figure 3: Processing insertions with DDE labels

with label 1.4.1, we get its label 1.4.2 by adding 1 to the local order of 1.4.1. Similarly, the label of node D is 1.4.3. Node E is inserted between two nodes with labels 1.2.1 and 1.2.2 and its label is 2.4.3 which equals to 1.2.2+1.2.3. Likewise, the labels of node F and G are 3.6.5 (2.4.3+1.2.2) and 5.10.8 (2.4.3+3.6.5) respectively. Next, node H is inserted as the child of leaf node 1.2.1, its label is 1.2.1.1 which is concatenation of its parent's label and 1. Node I is also inserted below a leaf node 3.6.5 and its label is therefore 3.6.5.1.

Among the insertions shown Figure 3, we consider the correctness of the following special cases obvious because the resulting labels are almost the same as the initial labeling, so proofs are ignored here.

- **Leftmost insertion.** When a new node is inserted before node $A : a_1.a_2 \dots a_n$ where A is the first child of a node, we assign label $a_1.a_2 \dots (a_n - 1)$ to this node.
- **Rightmost insertion.** When a new node is inserted after node $A : a_1.a_2 \dots a_n$ where A is the last child of a node, we assign label $a_1.a_2 \dots (a_n + 1)$ to this node.
- **Insertion below a leaf node.** When a new node is inserted below a leaf node $A : a_1.a_2 \dots a_n$, we assign label $a_1.a_2 \dots a_n.1$ to this node.

In general, insertions can be made between any two consecutive siblings.

- **Insertion between two consecutive siblings.** When a new node is inserted between two consecutive siblings with labels A and B, we assign label $A + B$ to this node.

We prove the correctness of this case in Section 3.5.1. In conclusion, DDE labeling scheme supports insertions at arbitrary positions in an XML tree.

3.5.1 Correctness

Before we prove the correctness of DDE insertions, we introduce the following properties of DDE labels.

THEOREM 3.4 (TRANSITIVITY OF DDE ORDER). *Given three DDE labels A, B and C, such that $A \prec_{dde} B$ and $B \prec_{dde} C$, then $A \prec_{dde} C$.*

PROOF. To prove $A \prec_{dde} C$, we have to show that $A \preceq_{dde} C$ and $A \neq_e C$. From $A \prec_{dde} B$ and $B \prec_{dde} C$, we have $A \preceq_{dde} B$ and $B \preceq_{dde} C$, but $A \neq_e B$ and $B \neq_e C$. Since $\preceq_{dde}$ is transitive, $A \preceq_{dde} B$ and $B \preceq_{dde} C$ imply that $A \preceq_{dde} C$. The remaining problem is to prove $A \neq_e C$. Suppose $A =_e C$, we have $C \preceq_{dde} A$ which, together with $A \preceq_{dde} B$, imply $C \preceq_{dde} B$. Since we already have $B \preceq_{dde} C$, it follows that $B =_e C$. We have a contradiction here because $B \neq_e C$. □

LEMMA 3.4 (SYMMETRY OF SIBLING RELATIONSHIP). *If $A : a_1.a_2 \dots a_m$ is a sibling of $B : b_1.b_2 \dots b_m$, then B is a sibling of A.*

PROOF. Since A is a sibling of B, $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \dots = \frac{a_{m-1}}{b_{m-1}} = \beta$. Equivalently, $\frac{b_1}{a_1} = \frac{b_2}{a_2} = \dots = \frac{b_{m-1}}{a_{m-1}} = \frac{1}{\beta}$. Thus, B is a sibling of A. □

LEMMA 3.5 (TRANSITIVITY OF SIBLING RELATIONSHIP). *If $A : a_1.a_2 \dots a_m$ is a sibling of $B : b_1.b_2 \dots b_m$ and B is a sibling of $C : c_1.c_2 \dots c_m$, then A is a sibling of C.*

PROOF. Since A is a sibling of B, $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \dots = \frac{a_{m-1}}{b_{m-1}} = \beta$, similarly, $\frac{b_1}{c_1} = \frac{b_2}{c_2} = \dots = \frac{b_{m-1}}{c_{m-1}} = \gamma$. $\frac{a_1}{c_1} = \frac{a_1}{b_1} \times \frac{b_1}{c_1} = \beta \times \gamma = \frac{a_2}{c_2} = \dots = \frac{a_{m-1}}{c_{m-1}}$. Thus, A is a sibling of C. □

LEMMA 3.6. *Let $A : a_1.a_2 \dots a_m$ be a sibling of of $B : b_1.b_2 \dots b_m$, then $C = A + B$ is a sibling of A and B.*

PROOF. Given that A is a sibling of B, we have $\frac{a_1}{b_1} = \frac{a_2}{b_2} = \dots = \frac{a_{m-1}}{b_{m-1}} = \beta$. As $\frac{a_1+b_1}{b_1} = \frac{a_1}{b_1} + 1 = \beta + 1$, it follows that $\frac{a_1+b_1}{b_1} = \frac{a_2+b_2}{b_2} = \dots = \frac{a_{m-1}+b_{m-1}}{b_{m-1}}$, so $A + B$ is a sibling of A. Consequently, $A + B$ is also a sibling of B from Lemma 3.5. □

LEMMA 3.7. *Suppose $A : a_1.a_2 \dots a_m$ and $B : b_1.b_2 \dots b_m$ are siblings, if $A' : a'_1.a'_2 \dots a'_n$ is an ancestor of A, then A' is an ancestor of $C = A + B$.*

PROOF. Since A' is an ancestor of A, we have $n < m$ and $\frac{a_1}{a'_1} = \frac{a_2}{a'_2} = \dots = \frac{a_n}{a'_n} = \beta$. From Lemma 3.6, $A + B$ is a sibling of A and therefore, $\frac{a_1+b_1}{a_1} = \frac{a_2+b_2}{a_2} = \dots = \frac{a_{m-1}+b_{m-1}}{a_{m-1}} = \gamma$. As a result, $\frac{a_1+b_1}{a'_1} = \frac{a_1+b_1}{a_1} \times \frac{a_1}{a'_1} = \beta \times \gamma = \frac{a_2+b_2}{a'_2} = \dots = \frac{a_n+b_n}{a'_n}$ and A' is an ancestor of C. □

LEMMA 3.8. *Given three DDE labels $A : a_1.a_2 \dots a_m$, $B : b_1.b_2 \dots b_m$ and $C = A+B$, such that A is a sibling of B and $A \prec_{dde} B$, if $A' : a'_1.a'_2 \dots a'_n$ is a child of A, then $A' \prec_{dde} C$.*

PROOF. Since A' is a child of A, we have $\frac{a_1}{a'_1} = \frac{a_2}{a'_2} = \dots = \frac{a_m}{a'_m}$ and equivalently $\frac{a'_1}{a_1} = \frac{a'_2}{a_2} = \dots = \frac{a'_m}{a_m}$. From Lemma 3.5 and Lemma 3.6, A is a sibling of $C = A+B$ and therefore, $\frac{a_1}{a_1+b_1} = \frac{a_2}{a_2+b_2} = \dots = \frac{a_{m-1}}{a_{m-1}+b_{m-1}}$. In addition, $A \prec_{dde} B$ implies that $a_m \times b_1 < b_m \times a_1$. Assuming $\frac{a'_1}{a_1} = \beta$ and $\frac{a_1}{a_1+b_1} = \gamma$, then $\frac{a'_1}{a_1+b_1} = \frac{a'_2}{a_2+b_2} = \dots = \frac{a'_{m-1}}{a_{m-1}+b_{m-1}} = \frac{a'_1}{a_1} \times \frac{a_1}{a_1+b_1} = \beta \times \gamma$. On the other hand, $a'_m \times (a_1 + b_1) = \frac{a_m \times a'_1}{a_1} \times (a_1 + b_1) = a_m \times a'_1 + \frac{a_m \times a'_1 \times b_1}{a_1} < a_m \times a'_1 + \frac{b_m \times a_1 \times a'_1}{a_1} = (a_m + b_m) \times a'_1$ and therefore, $A' \prec_{dde} C$. □

Now we are ready to prove the correctness of DDE insertion between two siblings which we state as the following theorem.

THEOREM 3.5. *To insert between two consecutive sibling nodes with DDE labels: A and B where $A \prec_{dde} B$, assigning $A + B$ to the new node is correct with respect to AD, PC, document order, sibling relationships and LCA computation.*

PROOF. We prove for each of the five properties:

P1 (AD RELATIONSHIP). AD relationship is preserved as all the ancestors of A and B are also ancestors of $A+B$ (Lemma 3.7).

P2 (PC RELATIONSHIP). PC relationship is also preserved because AD relationship is correct after insertions and level information of $A+B$ is correctly maintained, i.e. $|A+B| = |A| = |B|$.

P3 (DOCUMENT ORDER). Document order follows from $\prec_{dde}$ order. From Theorem 3.3, $A \prec_{dde} (A+B) \prec_{dde} B$. The set of nodes that precede $A+B$ should include A, all the nodes that precede A and all the descendants of A. Theorem 3.4 implies that any node which precedes A also precedes $A+B$. Moreover, any descendant of A should also precede $A+B$ from Lemma 3.8. Similarly we can show that the set of nodes that follow $A+B$ include B and all the nodes that follow B.

P4 (SIBLING RELATIONSHIP). From Lemma 3.6, $A+B$ is a sibling of A and B. Therefore $A+B$ is also a sibling of any sibling of A and B based on Lemma 3.5. On the other hand, Lemma 3.4 implies that A, B and all their siblings are also siblings of $A+B$.

P5 (LCA). LCA can be computed correctly because AD relationship is correctly maintained.

This concludes the correctness of DDE insertions. □

COROLLARY 3.1. *The set of DDE labels after arbitrary insertions is still inequivalent.*

Since document order is correctly maintained, the claim directly follows.

4. COMPACT DDE (CDDE)

In this section, we introduce a variant of DDE labeling scheme which we call Compact DDE (CDDE). CDDE is designed to enhance the performance of DDE for insertions.

4.1 Initial labeling

The label format of CDDE is the same as DDE which is a sequence of components separated by '.'. Moreover, the initial labeling of CDDE is the same as DDE (Figure 2), and is therefore the same as dewey. Unlike DDE labels whose first components are restricted to be positive decimal numbers, the first component of a CDDE label can be either positive or negative. We refer to the CDDE labels with positive first components as *positive CDDE labels* and those with negative first components as *negative CDDE labels*.

Let $A : a_1.a_2 \ldots a_m$ be a positive CDDE label, we refer to a_1 as *multiplier*, $a_1.a_2 \ldots a_{m-1}$ as *parent label* and a_m as *local order*. If $A : a_1.a_2 \ldots a_m$ is a negative CDDE label, then its multiplier, parent label and local order are a_1, $a_2.a_2 \ldots a_{m-1}$ and a_m respectively.

4.2 CDDE label to DDE label mapping

The properties of CDDE label, which include how various relationships can be established, are different from those of DDE. To simplify discussion, we take a shortcut by defining a mapping from CDDE label to DDE label.

Given a CDDE label $A : a_1.a_2.a_3 \ldots a_{m-1}.a_m$, we define a mapping $f^{cd} : CDDE\ label \rightarrow DDE\ label$ as:

$$f^{cd}(A) = \begin{cases} a_1.(a_1 \times a_2).(a_1 \times a_3) \ldots (a_1 \times a_{m-1}).a_m \\ \qquad when\ a_1 > 0 \\ (|a_1| \times a_2).(|a_1| \times a_3) \ldots (|a_1| \times a_{m-1}).a_m \\ \qquad when\ a_1 < 0 \end{cases}$$

Intuitively, the mapping is to apply the 'multiplier' to the parent label of the CDDE label. The multiplier is part of the parent label for positive CDDE labels, but is followed by the parent label for negative CDDE labels. For example, CDDE label 2.2.3 maps to DDE label 2.(2 × 2).3=2.4.3 whereas CDDE label -3.1.3.2.1 maps to DDE label (3×1).(3×3).(3×2).1=3.9.6.1.

Based on f^{cd} mapping, we define preorder (denoted as $\preceq_{cdde}$) on CDDE labels as:

DEFINITION 7 (PREORDER). *Given two CDDE labels A and B, $A \preceq_{cdde} B$ if and only if $f^{cd}(A) \preceq_{dde} f^{cd}(B)$.*

DEFINITION 8 (EQUIVALENCE RELATION). *Two CDDE labels A and B have equivalence relation if and only if $f^{cd}(A) =_e f^{cd}(B)$.*

Similarly, CDDE order (denoted as $\prec_{cdde}$) is defined as:

DEFINITION 9 (CDDE ORDER). *Given two CDDE labels A and B, $A \prec_{cdde} B$ if and only if $f^{cd}(A) \prec_{dde} f^{cd}(B)$.*

We summarize the properties of CDDE labels as:

- A CDDE label A is the parent/ ancestor/ sibling of another CDDE label B if and only if $f^{cd}(A)$ is the parent/ ancestor/ sibling of $f^{cd}(B)$.
- A CDDE label A precedes another CDDE label B in document order if and only if $A \prec_{cdde} B$.

4.2.1 Correctness of initial labeling

Given any CDDE label $A : 1.a_2.a_3 \ldots a_{m-1}.a_m$ in the initial labeling, we have $f^{cd}(A) = f^{cd}(1.a_2.a_3 \ldots a_{m-1}.a_m) = 1.a_2.a_3 \ldots a_{m-1}.a_m = A$, implying that the initial CDDE labels simply map those initial DDE labels. Therefore the correctness of CDDE initial labeling follows directly from that of DDE initial labeling which we have proved in Section 3.3.1.

4.3 CDDE label addition

Similar to DDE label addition, CDDE label addition applies to two CDDE labels with sibling relationship.

LEMMA 4.1. *Let $A : a_1.a_2.a_3 \ldots a_{m-1}.a_m$ and $B : b_1.b_2.b_3 \ldots b_{n-1}.b_n$ be two CDDE labels with sibling relationship, then a) a_1 and b_1 are both positive or both negative; b) $m = n$; and c) $a_2 = b_2, a_3 = b_3 \ldots a_{m-1} = b_{m-1}$.*

Lemma 4.1 obviously holds for the initial CDDE labels as they are all positive labels and among them, any two siblings have the same parent label. We will show that this lemma remains to be valid after updates in Section 4.4.

An important difference between CDDE and DDE is how insertions are handled. We define addition operation of CDDE labels as:

DEFINITION 10 (CDDE LABEL ADDITION). *Let $A : a_1.a_2.a_3 \ldots a_{m-1}.a_m$ and $A' : a'_1.a_2.a_3 \ldots a_{m-1}.a'_m$ be two CDDE labels with sibling relationship, addition of them is defined as:*

$$A +_c A' = (a_1 + a'_1).a_2.a_3 \ldots a_{m-1}.(a_m + a'_m)$$

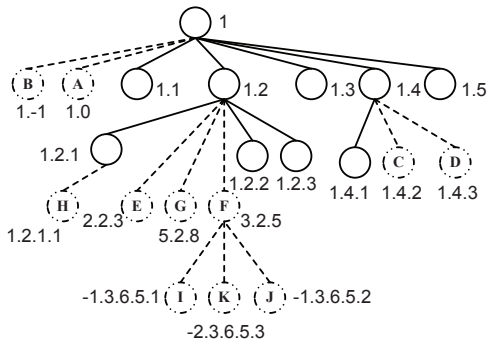


Figure 4: Processing insertions with CDDE labels

Different from DDE label addition, CDDE label addition only adds up the multipliers and local orders of two CDDE labels. As a result, the label size of CDDE increases at a slower rate than DDE after additions. However, the addition operations of DDE and CDDE labels are actually equivalent, as the following lemma implies.

LEMMA 4.2. *Given two CDDE labels* $A : a_1.a_2.a_3 \dots a_{m-1}.a_m$ *and* $A' : a'_1.a_2.a_3 \dots a_{m-1}.a'_m$.

$$f^{cd}(A +_c A') = f^{cd}(A) + f^{cd}(A')$$

PROOF. We consider two cases:

Both A and A' are positive CDDE labels. $f^{cd}(A +_c A') = f^{cd}((a_1 + a'_1).a_2.a_3 \dots (a_m + a'_m)) = (a_1 + a'_1).((a_1 + a'_1) \times a_2).((a_1 + a'_1) \times a_3) \dots ((a_1 + a'_1) \times a_{m-1}).(a_m + a'_m) = a_1.(a_1 \times a_2).(a_1 \times a_3) \dots (a_1 \times a_{m-1}).a_m + a'_1.(a'_1 \times a_2).(a'_1 \times a_3) \dots (a'_1 \times a_{m-1}).a'_m = f^{cd}(A) + f^{cd}(A')$

Both A and A' are negative CDDE labels. $f^{cd}(A +_c A') = f^{cd}((a_1 + a'_1).a_2.a_3 \dots (a_m + a'_m)) = (|(a_1 + a'_1)| \times a_2).(|(a_1 + a'_1)| \times a_3) \dots (|(a_1 + a'_1)| \times a_{m-1}).(a_m + a'_m) = (|a_1| \times a_2).(|a_1| \times a_3) \dots (|a_1| \times a_{m-1}).a_m + (|a'_1| \times a_2).(|a'_1| \times a_3) \dots (|a'_1| \times a_{m-1}).a'_m = f^{cd}(A) + f^{cd}(A')$

In both cases, $f^{cd}(A +_c A') = f^{cd}(A) + f^{cd}(A')$. □

LEMMA 4.3. *Suppose A and B are two CDDE labels such that $A \prec_{cdde} B$, then $A \prec_{cdde} (A +_c B) \prec_{cdde} B$.*

PROOF. Based on Definition 9, $A \prec_{cdde} B$ is equivalent to $f^{cd}(A) \prec_{dde} f^{cd}(B)$, which in turn implies that $f^{cd}(A) \prec_{dde} f^{cd}(A) + f^{cd}(B) \prec_{dde} f^{cd}(B)$ (Theorem 3.3). By Lemma 4.2, we can replace $f^{cd}(A) + f^{cd}(B)$ with $f^{cd}(A +_c B)$, which gives $f^{cd}(A) \prec_{dde} f^{cd}(A +_c B) \prec_{dde} f^{cd}(B)$. Thus, $A \prec_{cdde} (A +_c B) \prec_{cdde} B$. □

4.4 Processing updates

We illustrate how CDDE handles updates with an example.

EXAMPLE 4.1. *As illustrated in Figure 4, leftmost insertions (node A and B) and rightmost insertions (node C and D) are processed in the same way as DDE labels. However, when inserting between node 1.2.1 and 1.2.2, the new label for node E is 2.2.3* (1.2.1 +$_c$ 1.2.2). *Likewise, the labels for node F and G are 3.2.5* (2.2.3 +$_c$ 1.2.2) *and 5.2.8* (2.2.3 +$_c$ 3.2.5). *We postpone the discussion on the rest of the insertions to Section 4.4.1.*

Leftmost and rightmost insertions obviously do not violate the properties of sibling relationship stated in Lemma 4.1 as only local orders are changed. Moreover, we can see that Lemma 4.1 still holds after insertion between two positive CDDE labels (e.g. node E, F and G) because addition of CDDE labels only adds up the multipliers and local orders while their parent labels remain to be the same.

4.4.1 Processing insertion below a leaf node

We have shown how to process insertion below a leaf node with DDE label. The new label can be generated by concatenating the parent's label with 1. However, this method does not work for CDDE labels because it will produce new labels with incorrect parent labels. To accommodate such insertions, we introduce another operation which is used to get the label for the new node:

DEFINITION 11 (CDDE LABEL EXTENSION). *The extension operation of a CDDE label* $A : a_1.a_2.a_3 \dots a_{m-1}.a_m$ *is defined as:*

$$EXT(A) \rightarrow \begin{cases} -1.a_1.(a_1 \times a_2).(a_1 \times a_3) \dots (a_1 \times a_{m-1}).a_m.1 & \text{when } a_1 > 1 \\ a_1.a_2.a_3 \dots a_{m-1}.a_m.1 & \text{when } a_1 = 1 \\ -1.(|a_1| \times a_2).(|a_1| \times a_3) \dots (|a_1| \times a_{m-1}).a_m.1 & \text{when } a_1 < 0 \end{cases}$$

The next lemma shows the usefulness of CDDE label extension.

LEMMA 4.4. *Given a CDDE label* $A : a_1.a_2.a_3 \dots a_{m-1}.a_m$, $f^{cd}(EXT(A)) = f^{cd}(A).1$.

PROOF. There are three cases to be considered:

$\mathbf{a_1 > 1}$. $f^{cd}(EXT(A)) = f^{cd}(-1.a_1.(a_1 \times a_2).(a_1 \times a_3) \dots (a_1 \times a_{m-1}).a_m.1) = a_1.(a_1 \times a_2).(a_1 \times a_3) \dots (a_1 \times a_{m-1}).a_m.1 = f^{cd}(A).1$.

$\mathbf{a_1 = 1}$. $f^{cd}(EXT(A)) = f^{cd}(1.a_2.a_3 \dots a_{m-1}.a_m.1) = 1.a_2.a_3 \dots a_{m-1}.a_m.1 = f^{cd}(A).1$.

$\mathbf{a_1 < 0}$. $f^{cd}(EXT(A)) = f^{cd}(-1.(|a_1| \times a_2).(|a_1| \times a_3) \dots (|a_1| \times a_{m-1}).a_m.1) = (|a_1| \times a_2).(|a_1| \times a_3) \dots (|a_1| \times a_{m-1}).a_m.1 = f^{cd}(A).1$. □

When inserting a node below a leaf node with label A, we assign $EXT(A)$ to the new label.

EXAMPLE 4.2. *Consider the insertion of H in Figure 4, given that the parent of H has label 1.2.1, the label of H is* $EXT(1.2.1) = 1.2.1.1$. *Similaly, the label of I is* $EXT(F) = EXT(3.2.5) = -1.3.6.5.1$. *Inserting node J is just processed as a rightmost insertion and the new label is* $-1.3.6.5.2$. *To insert K between I and J, the new label is derived by adding the labels of I and J:* $-2.3.6.5.3$ ($-1.3.6.5.1 +_c -1.3.6.5.2$).

Lemma 4.1 still holds after insertion between two negative CDDE labels (e.g. node K) because only their multipliers and local orders are added up. The parent label of the new label remains the same as the parent labels of its left and right siblings.

4.4.2 Correctness

THEOREM 4.1. *To insert between two consecutive sibling nodes with CDDE labels: A and B where* $A \prec_{cdde} B$, *assigning* $A +_c B$ *to the new node is correct with respect to AD, PC, document order, sibling relationships and LCA computation.*

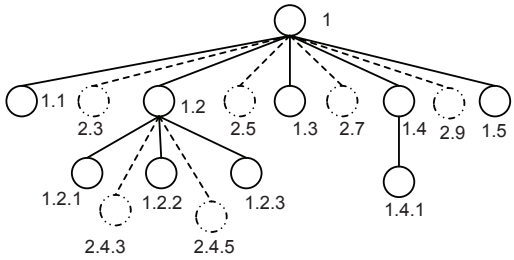


Figure 5: DDE labeling after uniform insertion

PROOF. Based on f^{cd} mapping, the properties of DDE labels can be adapted for CDDE labels. Moreover, it follows from Lemma 4.2 that the addition operations of DDE and CDDE labels are equivalent. Intuitively, assigning CDDE label $A +_c B$ to the new node is equivalent to the way that DDE labeling scheme handles insertion where the new DDE label is $f^{cd}(A) + f^{cd}(B)$. Thus, its correctness becomes the immediate consequence of Theorem 3.5. □

5. RELATIONSHIP COMPUTATION

In this section, we address the issue of how the various relationships of DDE and CDDE labels can be computed efficiently.

5.1 DDE labels

We have shown that DDE order, along with other properties of DDE labels, are generalized forms of dewey order and other properties of dewey labels. Given two DDE labels $A : a_1.a_2 \dots a_m$ and $B : b_1.b_2 \dots b_m$, they can be compared based on dewey order without any generalization if $a_1 = b_1 > 0$. Considering the fact that all the initial DDE labels start with 1, the chance that we have $a_1 = b_1 > 0$ is actually very high if the number of insertions is not too large or if the insertions are relatively uniform. As shown in Figure 5, if the insertions are performed uniformly between every two consecutive siblings, the new labels all have 2 as their first components. Moreover, since DDE labels can keep level information as their numbers of components after random updates, they are able to support fixed-cost computation of DDE order and other relationships even in the case of highly skewed insertions. In summary, the computation of various relationships is very efficient with DDE labels.

5.2 CDDE labels

The properties of CDDE, on the other hand, are defined by mapping CDDE labels to DDE labels. Therefore, it is natural to compute the various relationships between two CDDE labels by converting them to DDE labels. However, we will show that the conversion cost can actually be avoided from the following analysis.

LEMMA 5.1. *Assume A, B, A', B' are four DDE labels such that $A =_e A'$ and $B =_e B'$, then A is an ancestor of B if and only if A' is an ancestor of B'. The same result holds for PC, document order and sibling relationships. Let C be the LCA of A and B, C' be the LCA of A' and B', we have $C =_e C'$.*

Intuitively, it follows from Lemma 5.1 that any two DDE labels with equivalence relation are indeed *equivalent* in DDE labeling scheme. For example, we can replace a DDE label 2.4.6 with 1.2.3 ($2.4.6 =_e 1.2.3$), while not compromising the correctness of DDE labeling scheme. The proof is ignored here.

Given a CDDE label $A : a_1.a_2.a_3 \dots a_{m-1}.a_m$, we define a simple mapping $f^{scd} : CDDE\ label \rightarrow DDE\ label$:

$$f^{scd}(A) = \begin{cases} 1.a_2.a_3 \dots a_{m-1}.\frac{a_m}{a_1} & when\ a_1 > 0 \\ a_2.a_3 \dots a_{m-1}.\frac{a_m}{|a_1|} & when\ a_1 < 0 \end{cases}$$

For ease of exposition and simplicity, we allow a relaxed form of DDE labels where each component can be represented as a fraction of two decimal numbers. Note that the relaxed form is used for the purpose of comparison only.

LEMMA 5.2. *Let A be a CDDE label, we have $f^{cd}(A) =_e f^{scd}(A)$.*

PROOF. We consider the following two cases:

A is a positive CDDE label. $f^{cd}(A) = a_1.(a_1 \times a_2).(a_1 \times a_3) \dots (a_1 \times a_{m-1}).a_m$ and $f^{scd}(A) = 1.a_2.a_3 \dots a_{m-1}.\frac{a_m}{a_1}$. Since $\frac{a_1}{1} = \frac{a_1 \times a_2}{a_2} = \frac{a_1 \times a_3}{a_3} = \dots = \frac{a_1 \times a_{m-1}}{a_{m-1}} = \frac{a_m}{\frac{a_m}{a_1}} = a_1$, $f^{cd}(A) =_e f^{scd}(A)$.

A is a negative CDDE label. $f^{cd}(A) = (|a_1| \times a_2).(|a_1| \times a_3) \dots (|a_1| \times a_{m-1}).a_m$ and $f^{scd}(A) = a_2.a_3 \dots a_{m-1}.\frac{a_m}{|a_1|}$. Since $\frac{|a_1| \times a_2}{a_2} = \frac{|a_1| \times a_3}{a_3} = \dots = \frac{|a_1| \times a_{m-1}}{a_{m-1}} = \frac{a_m}{\frac{a_m}{|a_1|}} = |a_1|$, $f^{cd}(A) =_e f^{scd}(A)$. □

Lemma 5.1 and Lemma 5.2 together provide a very useful alternative for computing the relationships of CDDE labels. Give two CDDE labels A and B, their relationships can be computed based on $f^{scd}(A)$ and $f^{scd}(B)$ instead of $f^{cd}(A)$ and $f^{cd}(B)$.

How sibling relationships of CDDE labels can be computed directly is given in Lemma 4.1. Other optimizations are possible if we distinguish between positive and negative CDDE labels as the following lemmas illustrate:

LEMMA 5.3. *Suppose $A : a_1.a_2.a_3 \dots a_{m-1}.a_m$ and $B : b_1.b_2.b_3 \dots b_{n-1}.b_n$ are two positive CDDE labels, A is an ancestor of B if $m < n$, $a_2 = b_2, \dots a_{m-1} = b_{m-1}$ and $a_m = b_m \times a_1$.*

PROOF. Since A and B are positive CDDE labels, we have $f^{scd}(A) = 1.a_2.a_3 \dots a_{m-1}.\frac{a_m}{a_1}$ and $f^{scd}(B) = 1.b_2.b_3 \dots b_{n-1}.\frac{b_n}{b_1}$. A is an ancestor of B if $f^{scd}(A)$ is an ancestor of $f^{scd}(B)$, that is, $m < n$ and $\frac{1}{1} = \frac{a_2}{b_2} = \frac{a_3}{b_3} = \dots = \frac{a_{m-1}}{b_{m-1}} = \frac{\frac{a_m}{a_1}}{b_m}$. Therefore, we have $a_2 = b_2, \dots a_{m-1} = b_{m-1}$ and $a_m = b_m \times a_1$. □

LEMMA 5.4. *Suppose $A : a_1.a_2.a_3 \dots a_{m-1}.a_m$ and $B : b_1.b_2.b_3 \dots b_{n-1}.b_n$ are two negative CDDE labels, A is an ancestor of B if $m < n$, $\frac{a_2}{b_2} = \frac{a_3}{b_3} = \dots = \frac{a_{m-1}}{b_{m-1}} = \frac{a_m \times b_1}{b_m \times a_1}$.*

PROOF. Since A and B are negative CDDE labels, we have $f^{scd}(A) = a_2.a_3 \dots a_{m-1}.\frac{a_m}{a_1}$ and $f^{scd}(B) = b_2.b_3 \dots b_{n-1}.\frac{b_n}{b_1}$. A is an ancestor of B implies that $f^{scd}(A)$ is an ancestor of $f^{scd}(B)$ and therefore, $\frac{a_2}{b_2} = \frac{a_3}{b_3} = \dots = \frac{a_{m-1}}{b_{m-1}} = \frac{\frac{a_m}{a_1}}{\frac{b_m}{b_1}}$. Or equivalently, $\frac{a_2}{b_2} = \frac{a_3}{b_3} = \dots = \frac{a_{m-1}}{b_{m-1}} = \frac{a_m \times b_1}{b_m \times a_1}$. □

Similarly, we can compute other relationships based on f^{scd} mappings. The details are ignored here.

Dataset	Size (MB)	Total No. of nodes	Max/average fan-out	Max/average depth
XMark	113	1666315	25500/3242	12/6
Nasa	23.8	476646	2435/225	10/7
Treebank	85.4	2437666	56384/1623	36/8

Table 1: Test data sets

6. EXPERIMENTAL STUDY

6.1 Experimental setup

For the comparison between different labeling schemes, we consider the types of queries that they support as the primary factor. We focus on the comparison with dewey-based labeling scheme as they support efficient computation of sibling relationships and LCA, whereas containment labels do not have such properties. The comparison of range-based dynamic labeling schemes can be found in [9][15].

In addition, labeling schemes can be compared with a variety of measures including label size, label generation time and updating costs. We compare the labeling schemes proposed in this paper, i.e. DDE and CDDE, with ORDPATH and QED-Dewey (applying QED encoding scheme to dewey) as they both are dewey-based and can avoid re-labeling when updating the XML documents.

The evaluation of these labeling scheme was performed with XMark Benchmark[2], Nasa[1] and Treebank[1] data sets and their characteristics are shown in Table 1. All the experiments were conducted on a 2.33GHz dual-core PC with 4 GB of RAM.

6.2 Initial labeling

Initial labeling is evaluated by comparing ORDPATH and QED-Dewey labeling schemes against our DDE and the results are shown in Figure 6. CDDE is not shown here because its initial labeling is exactly the same as DDE. The initial labeling time for ORDPATH and DDE, as shown in Figure 6 (a), is approximately the same as their labels can be efficiently generated by scanning a document exactly once. However, the initial labeling time of QED-Dewey is much longer, as it needs to generate the dewey labels first which are then encoded into QED format. ORDPATH, as well as our DDE and CDDE are stored using the compressed ORDPATH format introduced in [11]. QED-Dewey is stored in its own physical storage format, with 0 as the separator between every two QED codes. For all the three data sets, we observe that DDE has the most compact initial label size, as illustrated in Figure 6 (b).

6.3 Querying static document

We test the query performance on all the three data sets. We present the results from Treebank data set as the other two data sets shown similar trends. Without any updates, the labels used for processing queries remain the same as the initial labels. We evaluate the query performance on initial labels by computing the most commonly used five relationships: document order, AD, PC, sibling and LCA. We choose the first 10000 labels from the initial labels of Treebank data set in document order and, for each pair of the labels, we compute all the five relationships. Note that as pointed out in [12], the LCA of a set of nodes is effectively the LCA of the first and the last node of the set in document order. Therefore we consider computing the LCA of two labels as a common function instead of many labels.

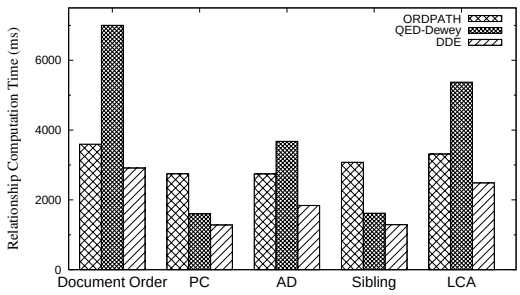


Figure 7: Time spent on computing different relationships

In Figure 7, we observe that the performance of different labeling schemes can be quite different for each of the five relationships. CDDE is not shown here because its performance is the same as DDE for static documents. While QED-Dewey is more efficient than ORDPATH for computing PC and sibling relationships, it is significantly slower for comparing document order and less efficient for AD relationship and LCA computation. For all the five relationships, our DDE outperforms ORDPATH and QED-Dewey.

6.4 Processing updates

6.4.1 Uniform Insertions

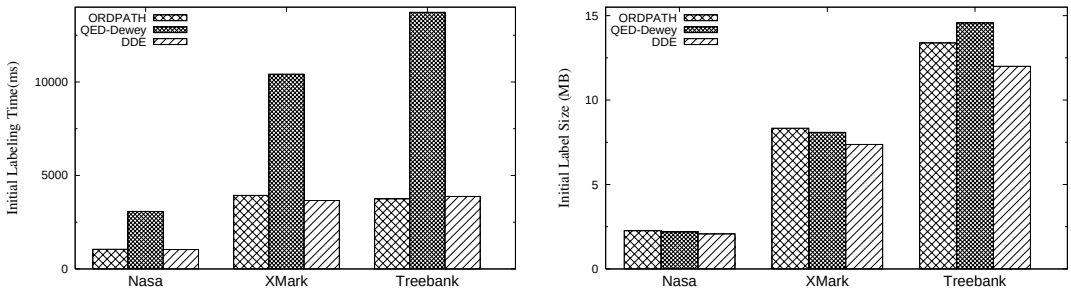
We test with insertions made uniformly between every two consecutive siblings. How these labeling schemes respond to uniform insertions is shown in Figure 8. The insertion time of ORDPATH is approximately the same as our DDE and CDDE whereas QED shows a slower updating time, as illustrated in Figure 8 (a). In Figure 8 (b), the comparison of label size after uniform insertions remains similar to that for the initial labels (Figure 6 (b)), with CDDE giving the most compact labels. The comparison of query performance, which we ignore here, is similar to that for the initial labels as well, since the quality of the labels is not much affected by uniform insertions.

6.4.2 Skewed Insertions

We classify skewed insertions into two different cases that are common in practice:

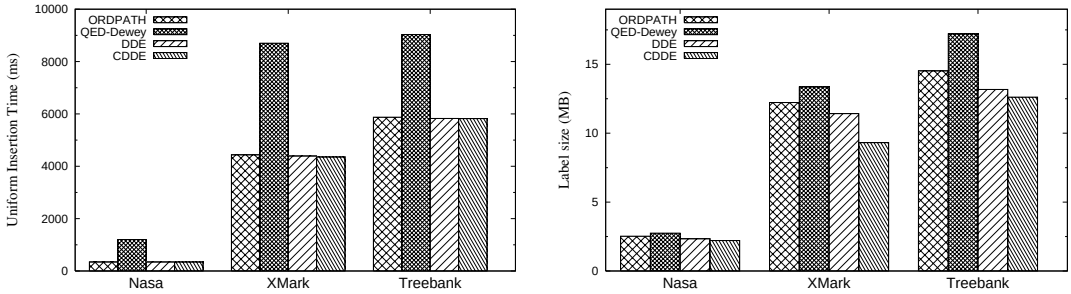
- **Ordered skewed insertion** refers to repeatedly inserting before or after a particular node.
- **Random skewed insertion** refers to repeatedly inserting between two nodes in random order.

Compared with uniform insertions, skewed insertions can have a more significant impact on the resulting qualities of labels. Figure 9 shows the updating cost and label size after ordered skewed insertions. The insertion time of ORDPATH, DDE and CDDE are negligible and their label sizes only increase slightly. In contrast, QED-Dewey has relatively higher updating time and its label size has shown a much higher increase. This result conforms to our previous discussions that the lengths of QED codes can increase at 1 or 2 bits per insertion in case of ordered skewed insertion, resulting in the fast increase of the overall label size. The



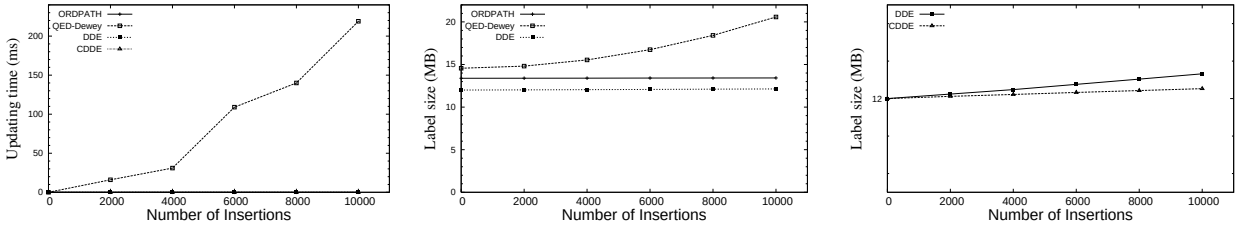
(a) Labeling time (b) Label size

Figure 6: Initial labeling



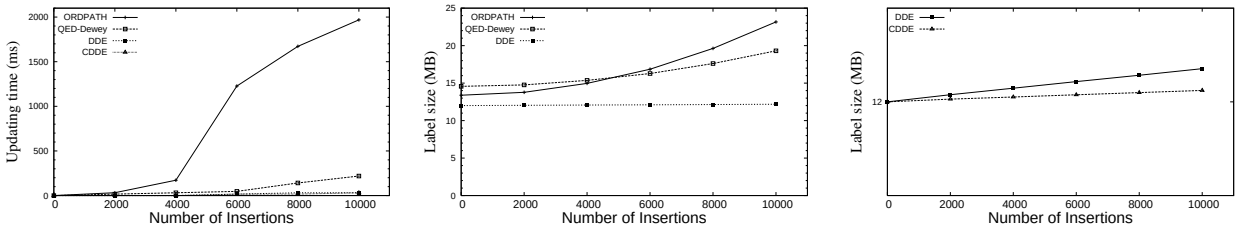
(a) Updating time (b) Label size

Figure 8: Uniform insertions



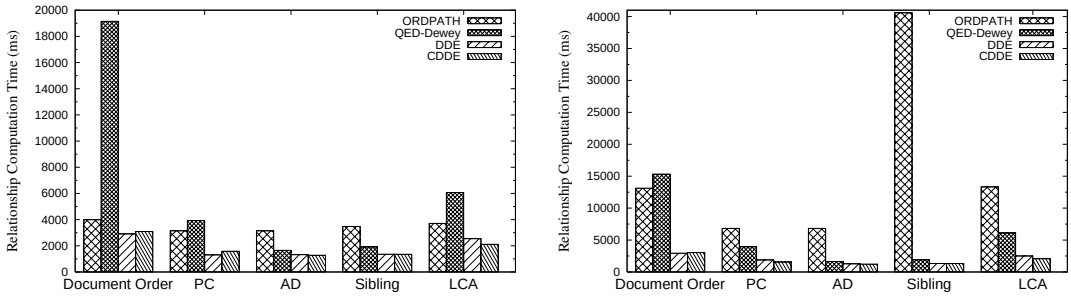
(a) Updating time (b) Label size (c) Label size

Figure 9: Ordered skewed insertions



(a) Updating time (b) Label size (c) Label size

Figure 10: Random skewed insertions



(a) Ordered skewed insertions (b) Random skewed insertions

Figure 11: Relationship computation time after skewed insertions

results for random skewed insertions are shown in Figure 10. The updating time and label size of ORDPATH increase at a much faster rate than the other labeling schemes. This is because random skewed insertions greatly increase the amount of 'caret's that are needed to be used in ORDPATH labels. For both types of insertions, our DDE and CDDE have shown the best performance in terms of updating time and label size. In addition, the label size of CDDE increases at a slower rate than DDE, which is what we have expected.

6.5 Querying dynamic document

To compare the query performance on dynamic XML documents, we adopt the same settings as the static case except the 10000 labels chosen include 2000 labels that are newly inserted. Figure 11 (a) gives the comparison of relationship computation time after ordered skewed insertions. Given the fast increase of QED-Dewey label size, it conforms to our expectation that its query response time also increases significantly, especially for document order. The comparison after random skewed insertions is shown in Figure 11 (b) where the query response time of ORDPATH increases significantly, particularly for sibling relationship. Nevertheless, our DDE and CDDE have demonstrated robust performance regardless of the order and number of insertions. Their query response times are least affected after both types of skewed insertions.

7. CONCLUSION

In this paper, we studied the problem of designing efficient labeling schemes for static and dynamic XML documents. We have presented a novel labeling scheme called DDE which not only achieves compact size and high query performance, but also completely avoids re-labeling when updating. A variant of DDE, namely CDDE has been introduced which is optimized for frequent insertions. Both DDE and CDDE have exhibited high resilience to skewed insertions in which case the qualities of existing labeling schemes degrade severely. Extensive experimental evaluation has demonstrated the benefits of our proposed labeling schemes over previous approaches.

8. REFERENCES

[1] University of Washington XML Repository. http://www.cs.washington.edu/research/xmldatasets/.

[2] XMark - An XML Benchmark Project. http://monetdb.cwi.nl/xml/downloads.html.

[3] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, and T. Rauhe. Compact Labeling Scheme for Ancestor Queries. *SIAM J. Comput.*, 2006.

[4] T. Amagasa, M. Yoshikawa, and S. Uemura. QRS: A Robust Numbering Scheme for XML Documents. *ICDE*, 2003.

[5] T. Bray, J. Paoli, C.M.Sperberg-McQueen, E.Maler, and F. Yergeau. Extensible Markup Language (XML) 1.0: fourth edition *W3C recommendation*, 2006.

[6] E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML Trees. In *SPDS*, 2002.

[7] C. Li and T. W. Ling. QED: A Novel Quaternary Encoding to Completely Avoid Re-labeling in XML Updates. In *CIKM*, 2005.

[8] C. Li, T. W. Ling, and M. Hu. Efficient Processing of Updates in Dynamic XML Data. In *ICDE*, 2006.

[9] C. Li, T. W. Ling, and M. Hu. Efficient Updates in Dynamic XML Data: from Binary String to Quaternary String. *VLDB J.*, 2008.

[10] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *VLDB*, 2001.

[11] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-friendly XML Node Labels. In *SIGMOD*, 2004.

[12] C. Sun, C.-Y. Chan, and A. K. Goenka. Multiway SLCA-based Keyword Search in XML Data. In *WWW*, 2007.

[13] I. Tatarinov, S. Viglas, K. S. Beyer, J. Shanmugasundaram, E. J. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *SIGMOD*, 2002.

[14] X. Wu, M. L. Lee, and W. Hsu. A Prime Number Labeling Scheme for Dynamic Ordered XML Trees. In *ICDE*, 2004.

[15] L. Xu, Z. Bao, and T. W. Ling. A Dynamic Labeling Scheme Using Vectors. In *DEXA*, 2007.

[16] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.

[17] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo, and G. M. Lohman. On Supporting Containment Queries in Relational Database Management Systems. In *SIGMOD*, 2001.