Labeling Dynamic XML Trees

Edith Cohen
AT&T Labs-Research
edith@research.att.com

Haim Kaplan
Tel Aviv University
haimk@post.tau.ac.il

Tova Milo
INRIA &Tel Aviv University
Tova.Milo@inria.fr

ABSTRACT

We present algorithms to label the nodes of an XML tree which is subject to insertions and deletions of nodes. The labeling is done such that (1) we label each node immediately when it is inserted and this label remains unchanged, and (2) from a pair of labels alone, we can decide whether one node is an ancestor of the other. This problem arises in the context of XML databases that support queries on the structure of the documents as well as on the changes made to the documents over time. We prove that our algorithms assign the shortest possible labels (up to a constant factor) which satisfy these requirements.

We also consider the same problem when "clues" that provide guarantees on possible future insertions are given together with newly inserted nodes. Such clues can be derived from the DTD or from statistics on similar XML trees. We present algorithms that use the clues to assign shorter labels. We also prove that the length of our labels is close to the minimum possible.

1. INTRODUCTION

XML is becoming the new standard for the exchange and publishing of data over the Internet [11, 1]. Documents obeying the XML standard can be viewed as trees, basically the parse tree of the document. XML database systems often give each item in the document (node in the tree) a unique logical identifier (called a *label*) and use those labels for an efficient processing of queries, in particular queries involving structural conditions or testing for changes in the document content:

Structural queries Typical queries over XML documents amount to finding nodes with particular tags (e.g. `book`, `author`, `price`) having certain ancestor relationship between them (e.g. book nodes that are ancestors of qualifying author and price nodes) [7, 12, 13, 1]. XML query engines often process such queries using an index structure, typically a big hash table, whose entries are the tag names and words in the indexed documents [15, 14, 9]. To allow structural queries, each node in the XML trees is given a unique label, and every entry (tag name or word) in the hash table is associated with the list of identifiers of the documents containing it, and for each such document the labels of the relevant nodes inside the document. The labels are designed such that given the labels of two nodes we can determine whether one node is an ancestor of the other. Thus structural queries can be answered using the index only, without access to the actual document.

Querying changes Users of XML data are often not only interested in querying the current values of documents but also in the changes in their content over time [15]. For example, they may be interested to know the price of a particular book in some previous time, or ask for the list of new books recently introduced into a catalog. To support such queries, XML databases, again, attach a unique label to each node in the tree and use it to connect and trace the various versions of a particular item throughout time.

Since XML documents found on the Web in general do not have identifiers for the various items, the database needs to provide the labels. Interestingly, all the systems that we are aware of use *two distinct labeling schemes* for the two tasks. An item is assigned one *persistent* label that does not change over time and is used to connect between versions, and another *structural* label (which might change when the document is updated) that reflects the ancestor relationships and is used for indexing. Queries involving both structural and historical conditions thus require going back and forth between the two labeling schemes; a significant overhead.

Why two labeling schemes? The reason is that all the structural labelings currently used by actual systems are designed for a static setting - the full structure of the document needs to be known before the labels can be chosen. When the structure changes the labels change as well. Thus to trace a given element across several document versions, a mapping between its different structural labels is required. To understand this, note that all these labelings are variants of the following *interval* scheme: number the leaves from left to right and label each node with a pair consisting of the numbers of its smallest and largest leaf descendants. An ancestor test then amounts to an interval containment test on the labels. Now, if the tree is updated and new leaves are added, the leaves need to be renumbered, and, consequently, the labels change. One may try to fix this by leaving some "gaps" between the numbers of the leaves. But if one part of the document is heavily updated we still may run out of available numbers and need relabeling.

The goal of the research reported in this paper is to design a *persistent structural labeling scheme*, namely a labeling scheme where labels encode ancestor relationships but need not be changed when the document is updated. The updates that we want to support are the addition and deletion of subtrees from the tree. (Naturally updates that move around existing subtrees cannot be supported with persistent labels since the existing ancestor relationships actually change).

Our abstraction of the problem is as follows. We consider a tree that is subject to insertions of nodes, s.t. when a node is inserted it must be a leaf (an insertion of a subtree can be modeled as a sequence of such insertions). When a node is inserted we have to assign it a label, and this label cannot be changed or replaced later on. Labels are assigned s.t. from the labels of two nodes one can determine if one is the ancestor of the other. We do not have an explicit *delete* operation in our abstract model since labels of deleted nodes cannot be reused. Deleted nodes still exist in some older version and a label should uniquely identify a node across all versions. For labeling purposes we might as well leave the deleted node in the tree and mark it with the version in which it ceased to exist. Thus the single tree in our abstraction represents the union of all versions in our previous discussion, and whenever we refer in the following to the *size of the tree* we actually count the *number of nodes inserted into the tree over time*, including ones that do not exist anymore in the most recent version.

Before presenting our results we should note that the length of the assigned labels is an important criteria in the quality of any such labeling scheme. This length determines the size of the index structure that contains the labels and thereby the feasibility of keeping this index in main memory. Consequently it is important to establish lower bounds on the length of the labels that any such scheme can produce, and design compact labelings that match those bounds. Observe that, depending on the physical representation used for the labels, labeling schemes may have different performance metrics: When fixed-size physical representation is used, the goal is to reduce the maximum length of a label. When variable-size representation is used, the sum of lengths of labels (or the average label length) needs to be reduced. In this paper we consider the maximum label length. We note, however, that the schemes we propose and the corresponding lower bounds are such that the average label length is typically within a small constant of the maximum. Thus, our results apply to both metrics.

Our results: We start by presenting a simple labeling scheme that uses $O(n)$-size labels, n being the size of the tree, and prove a matching lower bound of $\Omega(n)$ on the maximum label length of any possible labeling scheme. In contrast, in the static case where the complete tree is given in advance, there are known labeling schemes that use only $O(\log n)$-size labels (e.g. the simple interval scheme described above uses $2\log(n)$ bits). Thus, these results show an exponential performance gap between static and dynamic labeling.

The above worst-case analysis assumes no knowledge or restrictions on the final structure of the tree. A glance back at the real-life problem shows, however, that estimates on the shape or size of the XML trees (or of some particular subtrees) are often available. We show that these restrictions facilitate tighter analysis and can be integrated in the labeling scheme to reduce labels size.

The first issue that we consider is the depth and the width of the XML trees. By looking at about 2000 XML files collected by a crawler over the web [15] we observed that the average depth of an XML file is low, i.e. the trees are balanced with relatively high degrees. We thus analyzed the length of the labels as a function of the depth d of the tree and the maximal fan-out degree Δ of the nodes. We present a labeling algorithm that, even without knowing d and Δ in advance, generates labels of length bounded by $O(d \log \Delta)$. And again we also prove a matching lower bound of $\Omega(d \log \Delta)$.

The second issue that we consider is estimates on the size and shape of a subtree that will emerge underneath a particular node. Such estimates are often available in practice, and can be derived from the DTD of the XML file or from statistics of similar documents that obey the same DTD. We model this additional information by insertion sequences with *clues*. We then show that, even without any restriction on the actual depth or width of the trees, simple clues allow for much more efficient dynamic labelings. We define and analyze two types of clues. We first consider *subtree clues*, where with each inserted node v we are given an estimate, within a constant factor, on the final number of descendants of v. For this model, we provide tight upper and lower bounds of $\Theta(\log^2 n)$ on the length of the labels. Thus, subtree clues allow for considerably more efficient labels, but still do not match the performance of an off-line labeling of the tree. The second type of clues, *sibling clues*, are more informative and also include an estimate on the number of descendants of the future siblings of v. For insertion sequences with sibling clues we establish matching upper and lower bounds of $\Theta(\log n)$ on the maximum length of the labels. Thus, asymptotically, insertion sequences with sibling clues can be labeled on-line as well as they can be labeled off-line.

Finally, we also show how the above labeling algorithms can be extended to cope with wrong estimates, allowing persistent labeling even when the clue decelerations turn out to underestimate the actual size of the final subtrees.

We start with preliminary definitions in Section 2. In Section 3 we analyze insertion sequences without clues, and in Sections 4 and 5 insertions with clues. Section 6 deals with wrong clues. We conclude with related work.

2. PRELIMINARIES

We start by defining some of the basic terms used in the sequel.

A *static structural labeling scheme* is a pair, $\langle p, L\rangle$, where p is a 2-ary predicate over binary strings and L is a labeling function that given a tree T assigns a distinct binary string $L(v)$ for each node $v \in T$. The predicate p and the labeling function L are such that for every tree T and every two nodes $v, u \in T$, $p(L(v), L(u))$ evaluates to TRUE iff v is an ancestor of u.

A *persistent structural labeling scheme* is also a pair $\langle p, L\rangle$ where p, as before, is a binary predicate over strings. The labeling function L, however, rather than getting as input a full tree, gets a sequence of insertions of nodes into an initially empty tree. The root is the first to be inserted. Each subsequent insertion is of the form "insert node u as a child of node v". (So when u is inserted its parent v must already be in the tree). L does not know the insertions sequence in advance but receives them online. As each node is inserted, L assigns it a binary string. The label cannot be changed subsequently. The labeling L and the predicate p are such that for every insertion sequence and every two nodes v, u in the resulting tree, $p(L(v), L(u))$ evaluates to TRUE iff v is an ancestor of u.

The labeling function L can be deterministic or randomized, and the scheme will be called a deterministic/randomized, respectively.

Several static structural labeling schemes with short labels have been recently designed [2, 8, 4]. These schemes have been analyzed both theoretically and experimentally. (See Section 7 for further details.) In contrast our focus here is on the design of persistent structural labeling schemes. For this kind of labeling we are not aware of any previous work. Unless stated otherwise, the term *labeling scheme* in the sequel refers to a *persistent structural* one.

Two particular type of labels which we will use in the sequel are *range* and *prefix* labels.

- A range labeling comes equipped with some order relation $\leq$ overs binary strings. The label of a node v is interpreted as a pair of strings a_v, b_v and the predicate p is such that a node v is an ancestor of u iff $a_v \leq a_u \leq b_u \leq b_v$. The interval scheme described in the Introduction is an example of a static such labeling - a_v and b_v are interpreted as integers with $\leq$ being the standard order relation over integers.

- In a prefix labeling the predicate p is such that a node v is an ancestor of u iff $L(v)$ is a prefix of $L(u)$.

In the following sections we will see examples of these two types of schemes.

3. INSERTIONS WITHOUT CLUES

We start by considering arbitrary insertion sequences and suggest several simple labeling schemes for such sequences. We bound the lengths of the labels produced by these algorithms either in terms of the number of nodes in the resulting tree or in terms of the depth and degree of the tree. We also show that these algorithms are optimal.

The schemes presented in this section are all prefix schemes. Analogous range schemes can be developed using a technique presented in Section 6. To understand the main principal guiding our prefix schemes, lets look first at the static case, when the full tree is given in advance. Static prefix schemes typically work as follows. They assign to the outgoing edges of each node a set of prefix-free binary strings[1], and then, starting from the root and going down, define the label of each node to be the concatenation of its parent label and the string assigned to the edge leading to the node [8]. Consider, for example, a node v with three children v_1, v_2, v_3. We can assign the strings "0", "10", and "11" to the three edges (v, v_1), (v, v_2), and (v, v_3), respectively. So the labels of v_1, v_2, and v_3 are $L(v_1) = L(v) \cdot 0$, $L(v_2) = L(v) \cdot 10$, and $L(v_3) = L(v) \cdot 11$. The problem with using this scheme in a dynamic setting is that if the tree changes, e.g. we add a new child v_4 to v, there is no string that we can attach to the new edge (v, v_4). This is because any string would have one of the strings 0, 10 , and 11 as a prefix. The solution, which is the basis of all the persistent prefix schemes presented in the rest of this section, is to refrain from utilizing all possible prefixes. We always make sure that for every node v we can extend the currently assigned strings to the edges outgoing of v to a larger prefix free collection.

Consider the following simple prefix labeling scheme. We label the root with the empty string. The first child of the root is labeled with "0", the second child with "10", the third with "110" (rather than the "11" in the above example), the forth with '1110", etc. Similarly for any node v the first child of v is labeled with $L(v)$."0", the second child of v is labeled with $L(v)$."10", the third with $L(v)$."110", and the i^{th} child with $L(v)$."$111^{i-1}0$".

It is easy to see that this is indeed a correct prefix scheme, namely for all pairs of nodes v, u, $L(v)$ is a prefix of $L(u)$ iff v is an ancestor of u. Also, by induction it is easy to prove that the length of the maximum label is at most $i - 1$ after inserting i nodes including the root. So for any n-node tree the maximum label length is at most $n - 1$. This without any need to know n in advance.

Interestingly, the following theorem shows that no labeling scheme (regardless if it is prefix based, range based, or uses any other labeling type) can achieve better bound on the labels length.

THEOREM 3.1. *For every deterministic labeling scheme $S = \langle p, L\rangle$ there is an insertion sequence of length n such that S assigns a label of length at least $n - 1$ for some node in the sequence.*

PROOF. For a labeling scheme S we define $L(S, n)$ to be the set consisting of all labels that S uses to label insertion sequences of length n. We also define $P(n)$ to be the minimum over all labeling schemes of $|L(S, n)|$.

We claim that $P(n)$ satisfies the recurrence $P(n) \geq 2 * P(n-1)$, $P(1) = 1$. Therefore it follows that $P(n) \geq 2^{n-1}$ from

[1]A set of strings is *prefix-free* if no string in the set is a prefix of another.

which the theorem follows. To prove the claim consider an arbitrary insertion sequence of n nodes. After inserting the root r and the first of its children, say v, then we can partition the set of all labels that S uses to label trees with n nodes into two disjoint sets. The first set contains all labels which can be used for descendants of v, together with the label of v itself. The second set consists of all labels which can be used for descendants of r that are not descendants of v together with the label of r. The size of the first set must be at least $P(n-1)$ since v can root an arbitrary tree with $n-1$ nodes. Similarly the size of the second set should also be at least $P(n-1)$. $\square$

The above proof assumes no restrictions on the tree structure. In particular it relies on the fact that a node can have an arbitrary number of children. For XML files, the DTD may restrict the number of children, e.g. bounding it by some constant Δ. It turns out, however, that this does not change asymptotically the situation: we can still prove the following slightly weaker lower bound.

THEOREM 3.2. *For every deterministic labeling scheme S and every constant Δ, there is an n-node insertion sequence constructing a tree of maximum degree Δ on which S assigns a label of length at least $n\log_2(1/\alpha)-O(1)$, where α is a root of $x+x^2+\dots x^{\Delta}=1$.*

In particular the theorem shows that even if we restrict ourself only to binary trees, still, any deterministic labeling scheme will have some label of size $\Omega(n)$, or, more precisely, of size at least $0.69n - O(1)$ (since $\alpha = 0.618..$ for $\Delta = 2$). We prove Theorem 3.2 by constructing a sequence in which each insertion decreases the remaining labels of the current "chosen node" by a factor of α on average. We omit the details of the proof from this extended abstract.

What happens if the depth of the tree is also restricted? By looking at about 2000 XML files collected by a crawler over the web [15] we observed that the average depth of an XML file is low, i.e. the trees are balanced with relatively high degrees. We thus tried to find a more suitable labeling scheme for such trees.

As before, the children of a node v have the label of v concatenated with the string attached to their incoming edge. The string $s(i)$ for the i^{th} child is defined such that

$$s(1), s(2), s(3), \dots = 0, 10, 1100, 1101, 1110, 11110000, \dots$$

Namely, to obtain $s(i+1)$ we increment the binary number represented by $s(i)$ and if the representation of $s(i)+1$ consists of all ones we also double its length by adding a sequence of zeros.

The heuristics guiding this scheme is that the more children that a node already has, the more likely for it to get additional children. So rather than allocating for the new child the shortest possible available prefix-free string (as done in the first scheme presented at the beginning of this subsection), we give it instead a longer one. This investment is likely to pay off as it will shorten the labels of forthcoming siblings. In the first scheme, for each new child, the length of the assigned prefix free string grows by exactly one bit. In contrast, here, the length may grow by several bits at once. But then can stay the same for several future coming nodes (until it needs again to grow).

How long are the labels obtained by this scheme? A careful analysis of the algorithm shows that for all i, $|s(i)| \leq 4\log(i)$, and therefore,

THEOREM 3.3. *The maximum length of a label using this scheme is at most $4d\log(\Delta)$, d being the maximal depth of the tree and Δ the maximum outdegree of a node.*

Our algorithm works correctly even if Δ and d are not known in advance. We can also show that the latter algorithm is optimal up to a constant factor. Indeed, a full tree of depth d and out-degree Δ has more than Δ^d nodes. Hence, just to be able to assign distinct labels to the nodes of such a tree, any labeling scheme will require labels of length $\geq d\log_2\Delta - 1$.

Can randomization help? A randomized labeling scheme selects the label of an inserted node according to some probability distribution. The following theorem extends the above lower bounds to randomized labeling schemes showing that randomization essentially cannot help.

THEOREM 3.4. *For any randomized labeling scheme there is an insertion sequence for which the expected length of the maximum label is at least $n/2-1$. This holds even if the out-degree of nodes is bounded by some $\Delta \geq 2$.*

The proof constructs a probability distribution on request sequences that causes every deterministic labeling scheme to perform bad on this probability distribution. Then from Yao's lemma [16] it follows that this bad performance holds also for randomized labeling. The details are omitted.

Furthermore notice that the $\Omega(d\log\Delta)$ lower bound for trees with bounded depth and degree applies to randomized schemes as well.

4. LABELING WITH A CLUE

We have seen that for arbitrary trees, any persistent labeling scheme would need labels of length $\Omega(n)$ for some inputs. In contrast, simple static labeling schemes guarantee maximum label length of $\Theta(\log n)$ on arbitrary trees (e.g. the interval scheme presented in the Introduction has labels of length $2\log(n)$). To understand this exponential gap better and find ways to avoid it, we consider in this section insertion sequences such that with each inserted node, the algorithm gets a small amount of additional information, which we call a *clue*.

Clues provided with the inserted nodes restrict the set of possible continuations of the sequence, and thereby the set of possible final trees. The labeling algorithms we consider here obtain as input a list of *insertions*, each insertion of a node v specifies a parent node under which v should be inserted and an accompanying clue. We obtain bounds on the performance of labeling with clues by first showing a tight relation between ancestor labelings and *integer markings* (which we define below). Then we establish tight bounds on integer markings.

We use the following notation. For an insertion sequence s and a node $v \in s$, let $\mathrm{pr}_s(v)$ denote the prefix of s up to (and not including) the insertion of v. We denote by $T_s(v)$ the tree defined by $\mathrm{pr}_s(v)$. We denote by $C_s(v)$ the set of complete insertion sequences, of the same length as s, that are "legal" continuations of $\mathrm{pr}_s(v)$. We also denote by $\tau_s(v)$ the set of all trees that can be obtained by such continuations of $\mathrm{pr}_s(v)$. Finally, for a node v we denote by $\mathrm{P}(v)$ the parent of v.

4.1 Integer marking and labeling schemes

An integer marking algorithm assigns to each inserted node v an integer $N(v) \geq 1$ such that, at the end of the insertion sequence, the following holds for every node v.

$$N(v) \geq \Sigma_{\{u \ \mid \ v=\mathrm{P}(u)\}} N(u) + 1 \ . \tag{1}$$

Any labeling algorithm A has a corresponding integer marking algorithm A_m as follows:

LEMMA 4.1. *Consider an insertion sequence s and the insertion of node $v \in s$, let $B(v)$ be the set of distinct labels assigned by A to a descendant of v (including v itself) over all insertion sequences in $C_s(v)$. Then A_m assigns the marking $N_A(v) = |B(v)|$.*[2]

PROOF. Let $u_1, u_2, \ldots$ be the children of v in the order they are inserted. Equation 1 holds since the sets $B(u_i)$ are disjoint[3]; $\forall i$, $B(u_i) \subset B(v)$; the label of v is in $B(v)$ but is not contained in any of the $B(u_i)$'s. □

As a corollary of Lemma 4.1 we obtain that the maximum length of a label assigned by A to a descendent of v in some tree $T \in \tau_s(v)$ is at least $\log N_A(v)$ (since $N_A(v)$ distinct labels must have at least one of length at least $\log N(v)$). Therefore by showing a lower bound on the minimum possible size of an integer marking one obtains a lower bound on the maximum label length of any labeling algorithm.

We next show how any integer marking algorithm can be converted to a labeling algorithm. Thereby we will be able to obtain a labeling algorithm by obtaining an integer marking algorithm. We consider both range and prefix schemes. First, given an integer marking algorithm we show how to obtain a range labeling algorithm. The length of the labels produced by the range labeling algorithm is at most $2(1 + \lfloor \log N(r) \rfloor)$. Next we show how to obtain a prefix labeling algorithm. The length of the labels produced by the prefix labeling algorithm is at most $\lceil \log N(r) \rceil + d$ where d is the depth of the final tree obtained at the end of the insertion sequence.

[2]We use the integer marking algorithm A_m only as an analytic tool, thus we do not consider its complexity, which could be prohibitive.

[3]otherwise, we obtain a contradiction to the ancestor relation of the labels, since if $B(u_i)$ intersects $B(u_j)$, $j > i$, the insertion sequence and node that obtains that label under u_j would also be considered a descendant of u_i.

Range scheme: The algorithm is a persistent variant of the interval scheme described in the Introduction. The root is labeled by the interval $[1, N(root)]$, and each additional inserted node v is assigned a subinterval that contains $N(v)$ integers from the interval of its parent. (Siblings intervals are disjoint and assigned consecutively). It is easy to see that this yields a correct labeling with labels of length at most $2(1 + \lfloor \log(N(root)) \rfloor)$ bits.

Prefix scheme: The root is labeled by the empty string. When the i^{th} child, u_i of a node v is inserted, it is labeled by the label of v concatenated with a string s_i, s.t. (i) $s_1, \ldots, s_i$ are prefix free, and (ii) $|s_i| = \lceil \log(N(v)/N(u_i)) \rceil$. The following is easy to verify.

THEOREM 4.1. *Let d be the depth of the tree obtained at the end of the insertion sequence.*
(1) A string s_i with the above properties always exists, and
(2) the size of the labels generated by this prefix scheme is bounded by $\log(N(root)) + d$.

PROOF. (sketch) To find the strings we use an auxiliary data structure, a full binary tree T of depth $\lceil \log(N(v)) \rceil$. We label the left (respectively, right) outgoing edge of each node in T by "0" (resp., "1"), and label each node by the concatenation of the symbols on the path from the root. Now, when u_i is inserted, we look for the left most node of depth $\lceil \log(N(v)/N(u_i) \rceil$ in T s.t. neither the node nor any of its ancestors or descendants is marked. Its label is returned as the required s_i and the node becomes marked. To conclude the proof of item (1) it remains to prove the correctness of the above algorithm. We omit this here.

Finally, to see that the size of the labels is bounded by $\log(N(root)) + d$, observe that the longest labels are those of the leaves. The length of the label of a leaf node u is the sum of the s_i strings on the path to it, namely, can be described by a formula of the form

$$\lceil \log(N(root)/N(v_{i_1}) \rceil + \lceil \log(N(v_{i_1})/N(v_{i_2}) \rceil + \ldots + \lceil \log(N(v_{i_{d-1}})/N(u) \rceil$$

$$\leq \log(N(root)) + d.$$

□

We will in fact use in the sequel a slightly weaker notion of *almost integer markings* and show that almost-integer-markings can also be converted efficiently to labeling algorithms. An *c-almost integer marking* is defined for a constant c and is a marking $N(v)$ such that $N(v)$ satisfies Equation (1) for $N(v) \geq c$; every node v such that $N(v) < c$ has at most c descendants; and for each descendant u of v, $N(u) \leq N(v)$.

An almost integer marking can be converted into a labeling algorithm as follows. Let L be one of the prefix schemes from Section 3.

- Each node with $N(v) \geq c$ is labeled as with exact integer markings. (This works the same for both prefix and range labels).
- For a node v with $N(v) < c$, let u be the closest ancestor of v such that $N(u) \geq c$. The label of v is the concatenation of the label of u with the prefix-based label of v, as defined by L, within the subtree rooted at u.

Observe that when the labeling scheme used for nodes with $N(v) \geq c$ is a prefix scheme, the above combined scheme is also a prefix scheme. Thus ancestor test amounts, as before, to testing if one label is a prefix of the other. When the labeling scheme of nodes with $N(v) \geq c$ is a range scheme, to test for ancestor relationship one first needs to "chop" out, and compare (via range containment), the first $2(1+\lfloor \log N(r) \rfloor)$ bits of the labels. If those turn out to be identical then we need to continue and compare the remaining bits (via a prefix test). Finally note that although the labels now are longer than with exact integer marking, (e.g. by $O(c)$ bits), since c is a constant we still have asymptotically the same bounds.

We next show how clues on the *size of the subtree* that might emerge under a node can be used to derive tight bounds on magnitude of integer markings, and thus bounds on the number of labels needed to label the subtree. Note the distinction between the *size of a subtree* and the *number of labels* needed to label it. Clues on the possible size of XML subtrees can be derived from the DTD of the XML file or from statistics of similar documents that obey the same DTD. In contrast, the number of labels needed to label such a subtree may be much larger than the subtree itself, since it has to account for all the various possible structures of a subtree of that size. (As demonstrated for instance in the proof of Theorem 3.1).

For simplicity we assume first that all the provided size estimations are indeed correct. In Section 6 we will see how the algorithms can be extended to cope with wrong estimations.

4.2 Size estimations

We consider two types of clues. The first, which we call *subtree clue* consists of an estimate, up to a constant factor, of the number of future descendants of the inserted node. The second, which we call *sibling clues* consists of the subtree clue together with an additional estimate, of the number of descendants of future siblings of the inserted node.

The precise definition of the *subtree clues* is as follows. Each inserted node v is provided with a *range* $[l(v), h(v)]$. We consider ranges s.t. for all v, $h(v) \leq \rho * l(v)$ for some fixed $\rho \geq 1$. We call such ranges *ρ-tight*. The range is interpreted as a declaration that the final subtree rooted at v (including v itself) would contain at least $l(v)$ and at most $h(v)$ nodes. When analyzing the performance of our labeling schemes we will consider only "legal" insertion sequences, that is, sequences where all the declarations are met by the final tree. (Wrong declarations are considered in Section 6).

It is easy to see that if $\rho = 1$ (i.e. the subtree size is known exactly), then the above two labeling schemes can be used with $N(v) = l(v)$, and will produce correct labels of length $2(1 + \lfloor \log(n) \rfloor)$ and $log(n) + d$, resp., n being the size of the final tree. Interestingly, we show in Section 5 that for $\rho > 1$ any labeling scheme requires $\Omega(\log^2 n)$ bits on some insertion sequences. We also describe a labeling algorithm that achieves this bound.

The second, and stronger, type of clues that we consider are *sibling clues*. With each inserted node v, we obtain two ρ-tight ranges. The first range is essentially a subtree clue, with the same interpretation as above. Namely, it estimates, within a factor of ρ, the final size of the subtree rooted at v. The second range $[\overline{l}(v), \overline{h}(v)]$ estimates, within a factor of ρ, the sum of the sizes of all subtrees rooted at future (not yet inserted) siblings of v. As before, we first consider only sequences where all declarations are fulfilled by the final tree. In particular, declarations must be consistent with previous declarations. We show below that these more detailed clues allow for more efficient labeling and prove in Section 5 matching lower and upper bounds of $\Theta(\log n)$ on the maximum label length. Note that the $\Theta(\log n)$ bound asymptotically matches the bound for static tree labelings.

4.3 Current ranges

Before presenting the results, let us first explain a bit more the interaction between the range declarations of different nodes, and what additional information one can draw from them. As nodes are inserted to the tree and further range declarations are made, the set of possible final trees narrow down.

Two useful notions in our analysis of subtree and sibling clues are the *current subtree range* and the *current future range* of a node v, denoted by $[l^*(v), h^*(v)]$ and $[\hat{l}(v), \hat{h}(v)]$, respectively. Current ranges are the narrowest possible ranges that are consistent with all legal completions of the tree. (Thus they change, as nodes are inserted and more decelerations are made).

Let T be the tree constructed by the insertions so far, and let τ be the set of trees that can be formed by possible legal completions of the insertion sequence. The lower bound $l^*(v)$ of v is the smallest size of a subtree rooted at v in a tree $T' \in \tau$. The upper bound $h^*(v)$ is the maximal size of such a subtree. The current subtree range of a node is a sub-interval of the subtree clue that was provided with the node. As nodes are inserted into the graph, the current subtree range of other nodes may become more restricted.

Similarly, the *current future range* of a node v, $[\hat{l}(v), \hat{h}(v)]$, is such that $\hat{l}(v)$ is the minimum and $\hat{h}(v)$ is the respective maximum, over $T' \in \tau$, of the number of descendants of future children of v (namely children of v in T' not yet appearing in T). As nodes are inserted into the graph, the upper bound on the future range may become smaller. The lower bound, however, may decrease or increase (more precisely, as we shall see, with subtree clues it can only decrease, but sibling clues may cause it to increase).

It is not hard to show that ranges in τ are always contiguous. That is, for any $l^*(v) < k < h^*(v)$ there is a tree $T' \in \tau$ s.t. v has a subtree of size k. A similar claim holds for current future ranges. and $\hat{l}(v) < k < \hat{h}(v)$.

The following lemma provides a computational definition of the current subtree and future range for insertion sequences with subtree clues. The proof is rather straightforward, and thus omitted.

LEMMA 4.2. *The lower bound of the current subtree range of a node v can be recursively computed (bottom up) as follows.*

$$l^*(v) = \max\left\{ l(v), 1 + \sum_{\{u|\mathrm{P}(u)=v\}} l^*(u) \right\} . \quad (2)$$

The upper bound of v's current subtree range can be recursively computed (top down) as follows.
For the root node r, $h^*(r) = h(r)$. For a node v with parent $\mathrm{P}(v)$,

$$h^*(v) = \min\left\{ h(v), h^*(\mathrm{P}(v)) - 1 - \sum_{\{u|u\neq v \wedge \mathrm{P}(u)=\mathrm{P}(v)\}} l^*(u) \right\} . \quad (3)$$

The lower and upper bounds of the current future range of v can be computed as follows:

$$\hat{l}(v) = l^*(v) - 1 - \sum_{\{u|\mathrm{P}(u)=v\}} l^*(u) \quad (4)$$

$$\hat{h}(v) = h^*(v) - 1 - \sum_{\{u|\mathrm{P}(u)=v\}} l^*(u) . \quad (5)$$

The lemma shows how the current and future ranges can be updated as new nodes are inserted: When the root is inserted we have $l^*(r) = l(r)$, $h^*(r) = h(r)$, $\hat{l}(r) = l^*(r) - 1$, and $\hat{h}(r) = h^*(r) - 1$. When a node u is inserted under a node v, we have $l^*(u) = l(u)$ and $h^*(u) = \min\{h(u), \hat{h}(v)\}$. (We thus can assume w.l.o.g. that $0 \leq l(u) \leq h(u) \leq \hat{h}(u)$.) The future range of the new node u is $\hat{l}(u) = l^*(u) - 1$ and $\hat{h}(u) = h^*(u) - 1$. The insertion of u requires updating the current future range of its parent: $\hat{l}(v) \leftarrow \max\{0, \hat{l}(v) - l(u)\}$. It may also result in updates to the current ranges of other nodes. If we had $l(u) > \hat{l}(v)$ when u was inserted, we need to increase $l^*(v)$ by $l(u) - \hat{l}(v)$. The updates (increases) on the lower bounds on the current subtree range are then propagated through ancestors of the inserted node according to Equation 2, and upper bounds on the current ranges are then updated (decreased) by propagating them down from all nodes with increased lower bounds, according to Equation 3. The future current ranges are then recomputed according to Equations 4 for all nodes with a child with a modified current range.

When sibling clues are provided, the computation and update process of current ranges is somewhat more involved, and is postponed to the full version of the paper.

EXAMPLE 4.1. *Consider first an insertion sequence with* subtree *clues, and let $\rho = 2$. We start with the empty tree. Assume that the first inserted (root) node u has declared subtree range of $[5, 10]$, and the second insertion is a child v of u with a subtree range of $[4, 8]$. It is easy to see that the current future range of u is $[0, 5]$: Clearly the subtrees rooted at future children of u cannot contain more than 5 nodes as the tree has maximum allowed number of 10 nodes including the root u and the subtree rooted at v (which will contain at least 4 nodes); On the other hand, it is possible there will be no additional children whatsoever (hence 0 nodes) since the complete tree may have only 5 nodes, and this is already satisfied by the root itself and the 4 nodes that must appear in v's subtree.*

Note that, with subtree clues, the current future range is not necessarily ρ-tight (e.g., it can be $[0, 5]$). Sibling clues will restrict this range so that the gap between the lower and upper bound is at most a factor of two. Thus, the current future range will be contained in one of the following intervals $[0, 0]$, $[1, 2]$, $[2, 4]$, and $[3, 5]$. As soon will become evident, this seemingly small difference is what makes much more efficient labelings possible for sibling clues.

The example also illustrates why we cannot simply take the upper bound of the root subtree range, (10 in the example), also as the bound on the required number of labels (and hence possibly infer a bound on the labels length) . To be ready for all possible completions of the tree, we must have at least 8 available labels for the subtree of v and 5 more for the potential future children subtrees. This, together with the label of the root itself, implies that the domain of labels must contain at least 14 labels.

For convenience in our analysis, we assume below that the declared subtree clues are always contained in the current future range of the parent node. That is, when a node u is inserted as a child of v, $0 \leq l(u) \leq h(u) \leq \hat{h}(v)$. For sibling clues, we assume that $0 \leq \overline{l}(u) \leq \overline{h}(u)$; that $\overline{l}(u) \geq \hat{l}(v) - h(u)$; and $\overline{h}(u) \leq \hat{h}(v) - l(u)$. Note that this assumption is made without loss of generality, as it is possible to automatically narrow down the declarations to be consistent with current ranges.

5. BOUNDS ON INSERTIONS WITH CLUES

We first summarize our results for insertion sequences with subtree and sibling clues, then we illustrate the proof techniques used to establish those results. We start with subtree clues.

THEOREM 5.1. *For any deterministic (or randomized) labeling algorithm for insertion sequences with ρ-tight subtree clues, there is an insertion sequence of n nodes on which the scheme assigns an (expected) maximum label of length at least $\Omega(\log^2(n))$. Furthermore there is a labeling scheme that labels each such insertion sequence with labels of length $O(\log^2(n))$. The hidden constant factor degrades as ρ increases.*

To prove the lower bound we will show below that any integer marking algorithm has a sequence of n insertions where the mark of the root node is $n^{\Omega(\log(n))}$, which implies that we need $\Omega(\log^2(n))$ bits to represent the labels. To prove the upper bound we will show that legal integer marking can be obtained with assignments where $N(v) = h(v)^{O(\log h(v))}$, where h(v) is the upper bound of the subtree clue of v.

For a fixed ρ, the above results provide tight upper and lower bounds of $\Theta(\log^2 n)$ on the length of the labels. This is considerably better than the $\Theta(n)$ bounds when no clues are available, but still does not match the performance of static labeling schemes. We will next see that insertion sequences with sibling clues have matching upper and lower bounds of $\Theta(\log n)$ on the maximum length of the labels. Thus, asymptotically, insertion sequences with sibling clues can be labeled online as well as they can be labeled off-line.

THEOREM 5.2. *Consider insertion sequences where nodes come with both subtree and sibling clues.*

1. *Let* $S(n) = n^{1/\log_2((\rho+1)/\rho)}$. *A integer marking algorithm that takes* $N(v) = S(n)$, *when* v*'s subtree clue (and thus current subtree range) is* $[a, n]$, *for* $a \geq n/\rho$, *is a correct marking.*
2. *Any deterministic or randomized integer marking algorithm has a sequence where* n *is an upper bound on the range of the root on which it assigns to the root an (expected) marking of*

$$\Omega(n^{1/\log_2((\rho+1)/\rho)}) .$$

To conclude this section we present the proof of Theorem 5.1. The proof technique of Theorem 5.2 follows similar lines and is omitted here.

The upper and lower bound proofs rely on bounding the function $P(n)$ defined as follows. $P(n)$ is the minimum, over all integer marking algorithms A, of the maximum over insertion sequences, of the marking $N(v)$ that A assigns to a node v with current subtree range upper bound of $h^*(v) = n$.

PROOF. **(Theorem 5.1 deterministic lower-bound)** We show that $P(n) \geq (\frac{n}{2\rho})^{\Omega(\log n/\log(2\rho/(\rho-1))}$. Let A be an arbitrary marking algorithm and consider an insertion sequence s (see Figure 1) that inserts a root $r = v_0$ with clue $[\frac{n}{\rho}, n]$, then inserts a child v_1 of v_0 with clue $[\frac{n}{\rho} - 1, n - \rho]$ and continue inserting a path of $\frac{n}{2\rho}$ nodes $v_0, \ldots, v_{\frac{n}{2\rho}-1}$ in a similar fashion where the clue of v_i is $[\frac{n}{\rho} - i, n - i\rho]$. It is easy to check that after inserting these nodes then the current future range of v_0 is $[0, n\frac{\rho-1}{\rho}]$, and the current future range of v_i, $i \leq \frac{n}{2\rho} - 1$ is $[0, (n - i\rho)\frac{\rho-1}{\rho}]$.

To facilitate possible future insertions under v_i, the markings must be such that

$$N_A(v_i) \geq 1 + N_A(v_{i+1}) + P((n - i\rho)\frac{\rho - 1}{\rho}) .$$

For $0 \leq i \leq \frac{n}{2\rho}$, we obtain that

$$N_A(v_i) \geq 1 + N_A(v_{i+1}) + P(\frac{n}{2}\frac{\rho - 1}{\rho}) .$$

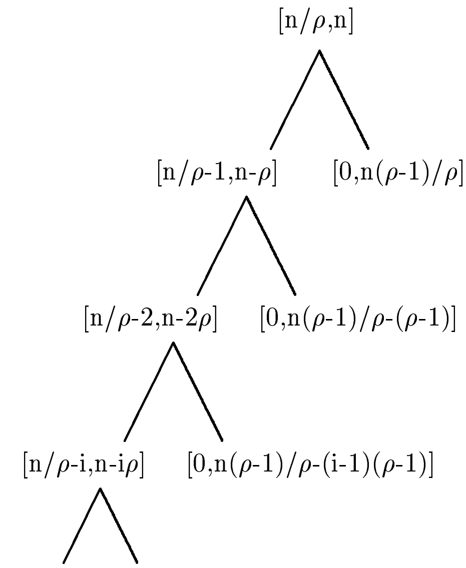

Figure 1: Insertion of a chain of descendants. The dotted lines are for current sibling ranges.

Therefore we obtain that $N(v_0) \geq \frac{n}{2\rho}P\left(\frac{n}{2}\frac{\rho-1}{\rho}\right)$. Since this holds for any marking algorithm A we obtain that $P(n)$ must satisfy the recurrence

$$P(n) \geq \frac{n}{2\rho}P\left(\frac{n}{2}\frac{\rho - 1}{\rho}\right) \geq \frac{n}{2\rho}\frac{n\frac{\rho-1}{2\rho}}{2\rho}\frac{n(\frac{\rho-1}{2\rho})^2}{2\rho}\cdots 1$$

where $P(1) = 1$. It follows that

$$P(n) \geq \left(\frac{n}{2\rho}\right)^{\Omega(\log n/\log(2\rho/(\rho-1))} .$$

□

PROOF. **(Theorem 5.1 randomized lower-bound)** We apply Yao's lemma which reduces the problem to finding a distribution over insertion sequences and showing that any deterministic algorithm is such that the expected length of the maximum-length label it assigns on these sequences is

$$M(n) = \left(\frac{n}{2\rho}\right)^{\Omega(\log n/\log(2\rho/(\rho-1))} .$$

We consider a distribution on insertion sequences obtained by the following process: Initially, the root is the "current node" and let n be as above. A chain of $n/(2\rho) - 1$ descendants is inserted starting from the current node, as described in the lower-bound proof of the deterministic case. The process then selects one of the $n/(2\rho) - 1$ nodes on the chain uniformly at random. The selected node becomes the new current node and a new chain insertion with $n \leftarrow n(\rho - 1)/(2\rho)$ is iteratively started from that node. The process continues until $n = 1$. The same calculation performed for the deterministic bound also shows that the number of distinct insertion sequences produced by this process is $M(n)$. Consider now a deterministic algorithm and the labels it assigns to nodes on these insertion sequences. Since the algorithm is deterministic, two insertion sequences with identical prefixes obtain identical labels for insertions that are part of the prefix. We claim, however, that labels assigned to nodes that are not part of the common prefix must be disjoint. It follows from this claim that the labels assigned to the last node of each insertion sequence are distinct. Since there are $M(n)$ such sequences, there are the same number of disjoint labels. Most labels in that set must thus be of length $\Omega(\log M(n))$; therefore the average length of these labels (over all these sequences must be $\Omega(\log M(n))$.

To prove the claim, consider the last chain-insertion that was common to both sequences. There are two different nodes on that chain, v_1 and v_2, such that the suffix of the first insertion sequences is all under v_1 and the suffix of the second sequence is all under v_2. Suppose w.l.o.g. that v_2 is a descendent of v_1. Assume to the contrary that a node u in the suffix of the first sequence had a common label to a node in the suffix of the second sequence. Since the label is on a suffix of the second sequence, it must be such that $L(u) \leq L(v_2)$ (in the ancestor relation sense), but the node u in the first sequence is not a descendant of v_2, and thus we obtain a contradiction. □

We use the following lemma for the upper-bound proof.

LEMMA 5.1. *Let* $f(x) = x^{\alpha \ln x}$ *and*

$$g(n,x) = f(x) + f(n - x/\rho - 2/\rho) \ .$$

Then

$$\forall\ 1 \leq x \leq n-1,\ g(n,n-1) \geq g(n,x) \ .$$

PROOF.

$$\frac{df(x)}{dx} = 2\frac{\alpha(\ln x)x^{\alpha \ln x}}{x}$$

using this we obtain that $dg(n,x)/dx$ is increasing. As $g(n,x)$ is continuous, the maximum point(s) must be obtained at the endpoints of the interval $[1, n-1]$, thus at $x = n-1$ or $x = 1$. It is not hard to see that $g(n, n-1) \geq g(n, 1)$. □

PROOF. **(Theorem 5.1 upper bound)** We prove an upper bound on $P(n)$. Consider a function $f(n)$ that satisfies the following inequalities for all $n \geq c(\rho)$, where $c(\rho)$ is a constant that depends on ρ.

$$f(n) \geq \max_{x\in[1,n]} \left\{ f(x-1) + f\left(n - 1 - \left\lceil \frac{x}{\rho} \right\rceil\right) + 1 \right\} \ , \quad (6)$$

and $f(n) = 0$ for $n \leq 0$.

We use two claims to conclude the proof.

- **Claim 1** states that $N(v) = f(h^*(v))$ constitute correct ($c(\rho)$-almost) integer marking (that is, satisfies Equation (1) for all legal insertion sequences). Thus, $f(n) \geq P(n)$.
- **Claim 2** states that the function

$$s(n) = (n/\rho)^{\log n/\log(\rho/(\rho-1))} \quad (7)$$

satisfies Equation (6).

To prove claim 1, we start by defining the unused marking $R(v)$ of a node: When v is inserted then $R(v) = f(h^*(v)) - 1$; When a child u is inserted under v then $R(v) \leftarrow R(v) - 1 - f(h^*(u))$. Note that a sufficient condition for (1) to hold is that we always have $R(v) \geq f(\hat{h}(v))$.

We now prove inductively that $R(v) \geq \hat{h}(v)$. First recall that the current range can only become more restricted as nodes are inserted elsewhere in the tree whereas $R(v)$ is decreased only when new children are inserted to v. Thus, it suffices to show that the inequality holds whenever a new child is inserted to v. Initially, when the node v is inserted then $R(v) = f(\hat{h}(v))$. Suppose that a child u is inserted to v. By the induction hypothesis, before the insertion v had current future range of $[l_a, h_a]$ and $R(v) \geq f(h_a)$. By definition of current future range, the current subtree range of u must be such that $h^*(u) \leq h_a - 1$. The updated current future range of v after the insertion has $\hat{h}(v) = h_a - 1 - l^*(u) \leq h_a - 1 - h^*(u)/\rho$. It follows from property (6) of $f()$ that $f(h_a) - 1 - f(h^*(u)) \geq f(\hat{h}(v))$. On the other hand, the unused marking of v after u is inserted is $R'(v) = R(v) - 1 - f(h^*(u))$. From these last two inequalities we obtain $R'(v) \geq f(\hat{h}(v))$.

The following two statements establish the correctness of claim 2.

1. $\max_{x\in[1,n]} \left\{ s(x-1) + s\left(n - 1 - \lceil \frac{x}{\rho} \rceil\right) + 1 \right\}$ is obtained for $x = n$.
2. We show that for $n \geq c(\rho)$

$$s(n) \geq s(n-1) + s(n\frac{\rho-1}{\rho}) + 1 \ . \quad (8)$$

The first statement is proven in Lemma 5.1. To prove (8), first note that

$$s(n) = \frac{n}{\rho} s\left(n\frac{\rho-1}{\rho}\right) \ . \quad (9)$$

Substituting using Equation (9) in inequality (8) we obtain that inequality (8) holds if for $n \geq c(\rho)$

$$\frac{n}{\rho} s\left(n\frac{\rho-1}{\rho}\right) \geq \frac{n-1}{\rho} s\left((n-1)\frac{\rho-1}{\rho}\right) + s\left(n\frac{\rho-1}{\rho}\right) + 1 \ . \quad (10)$$

Inequality (10) is equivalent to

$$\left(\frac{n}{\rho} - 1\right) s\left(n\frac{\rho-1}{\rho}\right) \geq \frac{n-1}{\rho} s\left((n-1)\frac{\rho-1}{\rho}\right) + 1 \ . \quad (11)$$

Substituting for $s()$ using (7) we obtain that inequality (11) is equivalent to

$$\left(\frac{n}{\rho}-1\right)\left(n\frac{\rho-1}{\rho^2}\right)^{\log_{\rho/(\rho-1)} n-1} \geq$$
$$\left(\frac{n-1}{\rho}\right)\left((n-1)\frac{\rho-1}{\rho^2}\right)^{\log_{\rho/(\rho-1)}((n-1)-1)}+1 \quad (12)$$

For $n \geq \rho^2/(\rho-1)+1$ the inequality

$$\left(\frac{n}{\rho}-1\right)\left(n\frac{\rho-1}{\rho^2}\right)^{\log_{\rho/(\rho-1)} n-1} \geq$$
$$\left(\frac{n-1}{\rho}\right)\left((n-1)\frac{\rho-1}{\rho^2}\right)^{\log_{\rho/(\rho-1)}(n-1)}+1\ , \quad (13)$$

which we obtained by substituting n for $n-1$ in the power on the right hand side of inequality (12), implies inequality (12). Dividing both sides of (13) by $x=\left(\frac{n}{\rho}-1\right)$ and $y=\left((n-1)\frac{\rho-1}{\rho^2}\right)^{\log_{\rho/(\rho-1)}(n-1)}$ we obtain that inequality (13) is equivalent to the following

$$\left(\frac{n}{n-1}\right)^{\log_{\rho/(\rho-1)} n-1} \geq \frac{n-1}{n-\rho}+\frac{1}{xy}\ . \quad (14)$$

For $n \geq \rho^2/(\rho-1)+1$, $y \geq 1$, so the inequality

$$\left(\frac{n}{n-1}\right)^{\log_{\rho/(\rho-1)} n-1} \geq \frac{n-1}{n-\rho}+\frac{1}{x} \quad (15)$$

implies that inequality (14) also holds. By substituting x and rearranging (15) we obtain that it is equivalent to

$$\left(1+\frac{1}{n-1}\right)^{\log_{\rho/(\rho-1)} n-1} \geq 1+\frac{2\rho-1}{n-\rho}\ . \quad (16)$$

If $\log_{\rho/(\rho-1)} n > 4\rho-1$ then

$$\left(1+\frac{1}{n-1}\right)^{\log_{\rho/(\rho-1)} n-1} \geq$$
$$\left(1+\frac{1}{n-1}\right)^{4\rho-2} \geq \left(1+\frac{4\rho-2}{n-1}\right) \quad (17)$$

and if $n > 2\rho-1$ then

$$\left(1+\frac{4\rho-2}{n-1}\right) \geq 1+\frac{2\rho-1}{n-\rho}\ . \quad (18)$$

Summarizing we obtain that for $n \geq c(\rho)$ where $c(\rho) = \max\{\rho^2/(\rho-1)+1, (\rho/(\rho-1))^{4\rho-1}, 2\rho-1\}$. the function s satisfies Inequality (8).

□

6. COPING WITH WRONG ESTIMATES

We briefly explain how the range and prefix labeling schemes presented in the previous section can be extended to cope with wrong estimates. First observe that over-estimation of the size of the final tree makes the labels longer than actually needed, but the labeling is still correct. We next show how to deal with under-estimations.

Extended range scheme: The key idea is to look at the binary code of the intervals assigned to a node and view the lower (upper) end points as virtually padded by an infinite sequence of 0s (1s). For example the range $[1001, 1101]$ is interpreted as $[10010000000\ldots, 11011111111\ldots]$. The order relation $\leq$ used here for determining interval inclusion is the lexicographical order on the (virtually padded) endpoint of the ranges (so the labels domain is virtually infinite). When a node v gets a new child v it needs to allocate to it a subrange that fits its clue declaration. If all declarations are correct, it should be able to do it within its (non padded) range. If it runs out of available space, the range is extended by using longer strings for the range endpoints, e.g. $[1101000, 1101111]$, and the required subrange within this extended range is allocated. Since $\leq$ is a lexicographical ordering on the virtually padded endpoints, the assigned range is still included within v's range. Further insertions may lead to further extensions, and so on.

Extended prefix scheme: We use here the same idea as in the prefix schemes of Section 3. Rather than consuming all the prefix-free strings, we do not assign the last string s_i, but, instead, use it as a basis for a longer string, say $s \cdot 0$. Then the following prefix-free assigned strings will be $s_i \cdot 10$, $s_i \cdot 110$, $s_i \cdot 1110$, $\ldots$.

It should be noted that the more wrong estimates are made, the longer the labels may be (up to $O(n)$ in the worst case). A related interesting open question is the design of optimal labeling schemes when clues are provided as distribution functions.

7. RELATED WORK

One can design labeling schemes in a static or dynamic setting. The static setting, where the full structure of the tree is known in advance, has been the focus of several recent works [2, 4, 17, 10, 8]. The best proposed schemes use labels of length $\log n + o(\log n)$ bits, and there is a matching lower bound of $\log n$ on the maximum label length of any such labeling scheme. None of these schemes, however, is suitable for a dynamic setting where the trees undergo changes through time.

Ancestor labeling schemes have also been studied in the context of object-oriented systems, as means to determine inheritance relationship among classes (see e.g. [5, 3]). The underlying graph describing the class hierarchy in these works is a DAG and the setting is static. Persistent object identifiers are used in object-oriented databases as means to identify individual objects throughout time. These ids however do not provide ancestor information [6].

The problem of designing a persistent labeling scheme for identifying nodes in a sequence of versions of an XML file has been recently studied by Marian et al [9], who suggested a scheme based on an inorder traversal of the original tree and the new inserted subtrees, with a relatively low storage overhead. Their labels, however, do not contain ancestor information and hence cannot be used for structural queries by a full text indexing mechanism. Marian et al [9] raised in their paper the question of whether an efficient persistent labeling scheme for multiple versions that also contains ancestor information is possible.

This question is addressed by the present paper. We modeled, analyzed, and obtained tight bounds for dynamic labeling of trees when the insertion sequence is accompanied with different levels of additional information. The bounds we obtained are summarized in the table below.

problem	static	dynamic		
		no clues	subtree clues	sibling clues
bounds	$\Theta(\log n)$	$\Theta(n)$	$\Theta(\log^2 n)$	$\Theta(\log n)$

Queries may sometimes need to test for parenthood, in addition to arbitrary ancestor relationships. This is often implemented, (and can be used with our labels as well), by attaching to each label the hight of the node in the tree - a node v is a parent of u iff the labels indicate ancestor relationship and the height of the first is smaller by one that that of the second[8, 15].

8. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan-Kaufmann, 340 Pine Street, Sixth Floor San Francisco, CA 94104, October 1999.

[2] S. Abiteboul, H. Kaplan, and T. Milo. Compact labeling schemes for ancestor queries. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January 2001.

[3] H. Ait-Kaci. An algebraic semantics approach to efficient resolution of type equations. *Theoretical Computer Science*, 45, 1986.

[4] S. Alstrup and T. Rauhe. Improved labeling scheme for ancestor queries. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (to appear) January 2002.

[5] Yves Caseau. Efficient handling of multiple inheritance hierarchies. In *OOPSLA*, pages 271–287, 1993.

[6] R. G. G. Cattell, editor. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Mateo, California, 1994.

[7] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0: An XML Query Language. W3C working draft, June 2001. `http://www.w3.org/TR/xquery`.

[8] H. Kaplan, T. Milo, and R. Shabo. A comparison of labeling schemes for ancestor queries. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, (to appear) January 2002.

[9] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-centric management of versions in an XML warehouse. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, September 2001.

[10] N. Santoro and R. Khatib. Labeling and implicit routing in networds. *The Computer J.*, 28:5–8, 1985.

[11] W3C. Extensible markup language (XML) 1.0. http://www.w3.org/TR/REC-xml.

[12] W3C. Extensible stylesheet language (XSL). http://www.w3.org/Style/XSL/.

[13] W3C. Xsl transformations (xslt) specification. http://www.w3.org/TR/WD-xslt.

[14] Xdex. `http://www.xmlindex.com`.

[15] Xyleme. A dynamic data warehouse for the XML data of the Web. http://www.xyleme.com.

[16] A. C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. 17th Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

[17] U. Zwick and M. Thorup. Compact routing schemes. In *Proceedings of th Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA*, July 2001.