SIAM J. COMPUT.
Vol. 39, No. 5, pp. 2048–2074
© 2010 Society for Industrial and Applied Mathematics

LABELING DYNAMIC XML TREES*

EDITH COHEN†, HAIM KAPLAN‡, AND TOVA MILO‡

Abstract. We consider online 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) each node is assigned a label immediately when it is inserted and this label remains unchanged, and (2) from a pair of labels alone, one 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 consider here the length of the assigned labels. We prove lower bounds on the length of labels which satisfy these requirements and provide labeling algorithms that match these bounds (up to a constant factor). 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/XML Schema 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.

Key words. XML, labeling schemes, ancestor queries, updates

AMS subject classifications. 68Q17, 68P15

DOI. 10.1137/070687633

1. Introduction.

XML is the de facto standard for the exchange and publishing of data over the Internet [18, 2]. 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 relationships between them (e.g., book nodes that are ancestors of qualifying author and price nodes) [7, 19, 20, 2]. XML query engines typically process such queries using an index structure, for instance a big hash table, whose entries are the tag names and words in the indexed documents [25, 23, 13]. 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. Node labels are used also in other types of XML index structures for similar purposes [21].

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 [25].

*Received by the editors April 9, 2007; accepted for publication (in revised form) November 12, 2009; published electronically February 24, 2010. A preliminary version of this paper appeared in *Proceedings of the Symposium on Principles of Database Systems* (PODS), 2002, pp. 271–281.
http://www.siam.org/journals/sicomp/39-5/68763.html
†AT&T Labs-Research, Florham Park, NJ 07932 (edith@research.att.com).
‡School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel (haimk@post.tau.ac.il, milo@cs.tau.ac.il). The second author's research was partially supported by the United States - Israel Binational Science Foundation, project 2006204, and by Israel Science Foundation grant 975-06. The third author's research was partially supported by the Israel Science Foundation.

For example, they may be interested to know the price of a particular book at some previous time or to 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 do not have identifiers for the various items, the database needs to provide such identifiers. Until 2002 (when the preliminary version of this paper appeared in PODS'02 [8]), all the systems we are aware of used *two distinct labeling schemes* for the two tasks. An item was 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 had to use the two labeling schemes, thereby suffering a significant overhead. The reason for using two labeling schemes was that all the structural labeling schemes existing at the time were designed for a static setting—the full structure of the document had to be known before the labels were assigned. 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 was required. To understand this, note that all these labelings were 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.[1]

The present work proposes a *persistent structural labeling scheme*, namely a labeling scheme where labels encode ancestor relationships but need not be changed when the document is updated. Before presenting our results we should note that the length of the assigned labels is an important criterion 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 (1) establish lower bounds on the length of the labels that any such scheme can produce and (2) design compact labeling that matches those bounds. We present in this paper such lower bounds and labeling schemes.

Following our work, there have been a variety of proposals for alternative labeling schemes with various properties (see section 11 for an overview). These labeling schemes are subject to the lower bounds we present here, and some can be combined with our algorithms to gain extra functionality. 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 provide upper and lower bounds on the maximum label length. We believe that on many families of trees, and corresponding insertion sequences, the performance of our algorithms would be better than their worst-case upper bounds.

The updates that we want to support are insertions and deletions of subtrees to/from the tree. (Updates that move around existing subtrees cannot be supported with persistent labels since the existing ancestor relationships actually change.) Our

[1] One may try to fix this by leaving some "gaps" between the numbers of leaves. But if one part of the document is heavily updated, we still may run out of available numbers and need relabeling.

abstraction of the problem is as follows. We consider a tree that is subject to insertions of nodes, such that 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 such that 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 versions, 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.

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 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 label size.

The first issue that we consider is the depth and the width of the XML trees. By looking at thousands of XML files collected by a crawler over the Web [6] it was 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 $O(d \log \Delta)$. 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/XML Schema of the XML file or from statistics of similar documents that obey the same DTD/Schema. 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 they still do not match the performance of an offline 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 online as well as offline.

Finally, we also show how the above labeling algorithms can be extended to cope with wrong estimates, allowing persistent labeling even when the clue declarations 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 4–9 insertions with clues. Section 10 explains how to deal with wrong clues. We conclude in section 11 with related work. As mentioned above, a preliminary version of this paper appeared in [8]. The present paper provides a comprehensive description of the proposed algorithms, the complexity analysis, and the proofs.

2. Preliminaries. We start by defining some of the basic terms used in what follows.

A *tree* in this paper will always be a rooted unordered tree. We denote by $p(v)$ the parent of a node v. The number of children of a node v is the *degree* of v. The *degree of a tree* T is the maximum degree of a node in T. The *depth* of a tree is the number of edges on the longest path from the root to a leaf. The *height* of a node v is the length (in edges) of the longest path from v to one of its leaf descendants. We denote the root of a tree by r. We denote by log the logarithm to the base 2, and we denote by $a \cdot b$ the concatenation of two strings a and b.

A *static structural labeling scheme* is a pair, $\langle P, L\rangle$, where P is a predicate over pairs of 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 if and only if 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 predicate over pairs of strings. The labeling function L, however, rather than getting as an input a full tree gets a sequence of insertions of nodes into an initially empty tree. The root is the first node 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.) The function L does not know the sequence of insertions in advance; it is in fact an algorithm that receives the insertions online. When we insert a node v, L assigns to it a binary string $L(v)$. This label cannot be changed subsequently. The labeling L and the predicate P are such that for every insertion sequence, and every two nodes v and u in the resulting tree, $P(L(v), L(u))$ evaluates to TRUE if and only if v is an ancestor of u.

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

Several static structural labeling schemes with short labels have been designed [1, 10]. These schemes have been analyzed both theoretically and experimentally. (See section 11 for further details.) In contrast our focus here is on the design of persistent structural labeling schemes. Unless stated otherwise, the term *labeling scheme* in what follows refers to a *persistent structural* one.

Two particular types of labels which we use in what follows are *interval* and *prefix* labels.

- An *interval labeling* comes equipped with some order relation $\leq$ over binary strings. The label of a node v is interpreted as a pair of strings a_v and b_v, and the predicate P is such that a node v is an ancestor of u if and only if $a_v \leq a_u \leq b_u \leq b_v$. That is, the interval $[a_v, b_v]$ associated with v contains the interval $[a_u, b_u]$ associated with u. The interval scheme described in the introduction is an example of a static interval labeling scheme—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 if and only if $L(v)$ 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 in terms of either the number of nodes in the resulting tree or in terms of the depth of the tree and/or the maximum degree of a node in the tree. We also show that these algorithms are optimal.

The schemes presented in this section are all prefix schemes. Analogous interval schemes can be developed using a technique presented in section 10. To understand the main principal guiding our prefix schemes, let us 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,[2] and then, starting from the root and going down, they 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 [10]. (The label of the root is the empty string.) 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. This way we make sure that we can extend the currently assigned strings to the edges outgoing from v to a larger prefix-free collection.

Consider the following simple prefix labeling scheme. We label the edge to the first child of a node v by 0, and the edge going to the second child by 10, and the edge to the third child by 110 (rather than 11 in the example above). So the label of the root is the empty string, and for any node v the first child of v is labeled with $L(v) \cdot 0$, the second child of v with $L(v) \cdot 10$, the third with $L(v) \cdot 110$, and the ith child with $L(v) \cdot 1^{i-1}0$.

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 $L(u)$ if and only if 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 is without any need to know n in advance.

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

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

Proof. We recursively define the following family $\mathcal{F}(n)$ of insertion sequences. The family $\mathcal{F}(1)$ consists of one sequence of two insertions: a root and a child of

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

the root. Assume we defined $\mathcal{F}(n-1)$. Let s be a sequence in $\mathcal{F}(n-1)$, and let r be the root of the tree that s generates. Note that the first insertion of s is of r. We have two sequences s_1 and s_2 in $\mathcal{F}(n)$ corresponding to s. Both s_1 and s_2 start with an insertion of a root r' and an insertion of a child w of r'. Then s_1 continues with the insertions in s starting from the second insertion with w playing the role of r in s. That is, a node inserted as a child of r in s is inserted as a child of w in s_1. The sequence s_2 continues with the insertions in s starting from the second insertion with r' playing the role of r in s. That is, a node inserted as a child of r in s is inserted as a child of r' in s_2. We call the sequences $\{s_1 \mid s \in \mathcal{F}(n-1)\}$ *type* 1 sequences and the sequences $\{s_2 \mid s \in \mathcal{F}(n-1)\}$ *type* 2 sequences.

It is easy to prove by induction that the number of sequences in $\mathcal{F}(n)$ is 2^{n-1} and each sequence is of length $n+1$.

We claim that the last insertion of each sequence in $\mathcal{F}(n)$ must get a different label by any correct labeling scheme. We prove this claim by induction. Assume that it is true for the sequences in $\mathcal{F}(n-1)$. Consider a labeling L of $\mathcal{F}(n)$. The restriction of L to sequences of type 1 induces a labeling of $\mathcal{F}(n-1)$. So for any two sequences s and s' in $\mathcal{F}(n)$ of type 1 the last node in s must get a label different from the label of the last node in s'. Similar argument shows that for any two sequences s and s' of type 2 in $\mathcal{F}(n)$ the last node in s must get a label different from the label of the last node in s'.

Let s_1 be a sequence of type 1, and let s_2 be a sequence of type 2. The second insertion both in s_1 and in s_2 is of the first child w of the root. Since at the point when w is inserted the algorithm cannot distinguish between s_1 and s_2, node w must get the same label both in s_1 and s_2. Let v be the last insertion of s_1, and let v' be the last insertion of s_2. For the labeling of s_1 to be correct the decoder seeing $L(v)$ and $L(w)$ should say that w is an ancestor of v. For the labeling of s_2 to be correct the predicate, P, upon seeing $L(v')$ and $L(w)$ should say that w and v' are unrelated. Therefore $L(v) \neq L(v')$.

So every algorithm needs (1) 2^{n-1} distinct labels to use for the last node in each sequence of $\mathcal{F}(n)$, (2) a label for the root, and (3) if $n > 1$ a label for the child of the root. Therefore the label on the worst case consists of n bits. Since sequences in $\mathcal{F}(n)$ are of length $n+1$ the lemma follows. □

The proof above 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 the situation asymptotically: 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 m-node insertion sequence constructing a tree of maximum degree Δ on which S assigns a label of length at least $m \log_2(\alpha) - O(1)$, where α is a root of $1 + x^2 + \cdots + x^{\Delta-1} = x^{\Delta}$.*

Proof. Define the following family $\mathcal{F}(n)$ of insertion sequences. If $n \leq \Delta$, then $\mathcal{F}(n)$ is defined as in the proof of Theorem 3.1. Otherwise the family $\mathcal{F}(n)$ consists of Δ types of sequences. A sequence s' in $\mathcal{F}(n)$ of type i, $1 \leq i \leq \Delta$, is obtained from a sequence s in $\mathcal{F}(n-i)$. Let r be the root of the tree produced by s. The sequence s' starts with an insertion of a root r', followed by i insertions of nodes $v_1, \ldots, v_i$, which are children of r'. Finally we add to s' the insertions of s, starting from the second, where each insertion in s of a child of r is modified to an insertion of a child of v_i.

An easy proof by induction shows that a sequence in $\mathcal{F}(n)$ is of length $n+1$.

From the proof of Theorem 3.1 it follows that for $n \leq \Delta$, $|\mathcal{F}(n)| = 2^n$. For $n > \Delta$ one can see that the number of sequences in $\mathcal{F}(n)$ satisfies the recurrence

$$|\mathcal{F}(n)| = |\mathcal{F}(n-1)| + \cdots + |\mathcal{F}(n-\Delta)| \ .$$

Solution to this recurrence is a linear combination of the roots of the polynomial $x^\Delta = x^{\Delta-1} + \cdots + 1$. So it is asymptotically equal to $c\alpha^n$, where α is the largest root of the polynomial $x^\Delta = x^{\Delta-1} + \cdots + 1$.

An argument similar to the one in the proof of Theorem 3.1 shows that the last insertion in each sequence in $\mathcal{F}(n)$ should have a different label. Indeed, this holds for sequences of the same type by induction. Let s_1 be a sequence of type i_1, and let s_2 be a sequence of type i_2, where $i_1 < i_2$. Let u_1 be the last node of s_1, and let u_2 be the last node of s_2. When u_{i_1} is inserted we still do not know whether the sequence is going to be s_1 or s_2. Therefore it will have the same label in both sequences. Since u_1 is a descendant of u_{i_1} but u_2 is not, the label of u_1 must be different from the label of u_2.

It follows that we need at least $c\alpha^n$ different labels, and therefore some labels must be of length $n \log_2(\alpha) - O(1)$ bits. □

In particular the theorem shows that even if we restrict ourselves 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 $\alpha \approx 0.69$ for $\Delta = 2$).

What happens if the depth of the tree is also restricted? By looking at thousands of XML files collected by a crawler over the Web [6] it was 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 better labeling scheme for such trees.

For $i \geq 1$ define a string $s(i)$ such that

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

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. It is straightforward to check that this collection of strings is prefix-free. For every node v we associate the string $s(i)$ with the edge from v to its ith child.

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 a longer one instead. 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 it can stay the same for several future nodes (until it needs again to grow). The following lemma bounds the length of the labels obtained by this algorithm.

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

Proof. Consider the length ℓ of the bit string $s(i)$. If we look at the second quarter of the ℓ bits from the left, then every possible bit string in this quarter is used by some $s(j)$, where $j \leq i$. So we have that $i \geq 2^{(\ell/4)}$ or $\ell \leq 4 \log(i)$. From this observation and the upper bound of Δ on the outdegree we obtain that the length of the string associated with every edge is bounded by $4 \log \Delta$. The lemma then follows from the upper bound on the depth of each 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 outdegree 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 $\geq d \log_2 \Delta$.

Can randomization help? A randomized labeling scheme selects the label of an inserted node according to some probability distribution. The following theorem extends our 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 of length m for which the expected length of the maximum label is at least $m-3$ bits. If the degree is bounded by Δ, then there is an insertion sequence for which the expected length of the maximum label is at least $m \log_2(\alpha) - O(1)$, where α is a root of $1 + x^2 + \cdots + x^{\Delta-1} = x^{\Delta}$.*

Proof. The first part follows by imposing a uniform distribution on the family $\mathcal{F}(n)$ of insertion sequences defined in the proof of Theorem 3.1. Recall that $|\mathcal{F}(n)| = 2^{n-1}$ and each sequence in $\mathcal{F}(n)$ is of length $n+1$. Consider the expected length of the maximum label assigned by a deterministic algorithm labeling a sequence from this uniform distribution.

Since each sequence contains a distinct label and the number of labels of less than y bits is smaller than 2^{y+1}, at most 2^{y+1} sequences can have a maximum label length smaller than y bits. It follows that the probability which we assign a label of length at least y is at least $(2^{n-1} - 2^{y+1})/2^{n-1} = 1 - (1/2^{n-y-2})$. Summing for $y = 0, 1, \ldots, n-3$ we get that the expected maximum label length is at least

$$\left(1 - \frac{1}{2^{n-2}}\right) + \left(1 - \frac{1}{2^{n-3}}\right) + \cdots + \left(1 - \frac{1}{2}\right) \geq (n-2) - \left(\frac{1}{2} + \cdots + \frac{1}{2^{n-2}}\right) \geq n-3.$$

The statement then follows by Yao's principle [26], which says that the best performance of a randomized algorithm is equal to the performance of the best deterministic algorithm on any particular probability distribution.

The second part of the theorem follows by applying an argument similar to the family of sequences in the proof of Theorem 3.2. □

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 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 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* $[\ell(v), h(v)]$. We consider ranges such that for all v, $h(v) \leq \rho * \ell(v)$ 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 $\ell(v)$ and at most $h(v)$ nodes. To simplify the presentation we do not require that $\ell(v)$ and $h(v)$ would be integers. The range $[\ell(v), h(v)]$ is equivalent to $[\lceil \ell(v) \rceil, \lfloor h(v) \rfloor]$. Notice that if $\rho = 1$, then $\ell(v) = h(v)$ is the exact size of the subtree which is rooted by v at the end of the insertion sequence.

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 $[\ell(v), h(v)]$ is a subtree clue which estimates to within a factor of ρ, the final size of the subtree rooted at v. The second range $[\overline{\ell}(v), \overline{h}(v)]$ estimates, within a factor of ρ, the sum of the sizes of all subtrees rooted at future (not yet inserted) siblings of v. Again we allow $\overline{\ell}(v)$ and $\overline{h}(v)$ to not be integers.

Clues given to different nodes are related to each other and may even be inconsistent. For example consider an insertion sequence that starts with an insertion of a root, r, with a subtree clue of $[3, 6]$ and continues with an insertion of a child v of r with a subtree clue of $[4, 8]$. These clues are inconsistent in the sense that there are points in the range of v that cannot be realized because, if they do, we violate the subtree clue of r. Specifically, node v cannot root a subtree with 8 nodes, as if that would be the case, then node r roots a subtree with at least 9 nodes, whereas the clue of r constrained the size of its subtree to be at most 6.

We define the subtree clues to be *consistent* if for every node v, when v is inserted, every point in its clue can be realized by some continuation of the sequence without violating any clue of a previously inserted node. For example if we start with a root r with a clue of $[3, 6]$ and then insert a child v of r, then $h(v)$ must be at most 5 since otherwise we will have a point in the clue of v that cannot be realized. However, the clue of v can be $[4, 5]$, although that makes the point 3 in the clue of r impossible to realize without violating the clue of v. Our definition of consistency requires only that any point in the range of the currently inserted node can be realized. In general as we insert nodes their clues would restrict further and further the possible ranges of previously inserted nodes.

4.1. Maintaining current ranges. As we observed, the insertion of a node v with a clue not only restricts the size of the subtree that is rooted by v in the final tree but also may tighten the constraint on the size of the subtrees rooted by other, already inserted, nodes. As another example assume we insert the root r with a subtree clue of $[3, 6]$. Then we insert a child, v_1, of r with a clue of $[2, 4]$ and subsequently a second child, v_2, of r with a clue of $[2, 3]$. At this point we already know that the tree must be of size at least 5. So the possible range of r narrows down to $[5, 6]$. Furthermore when v_2 is inserted its clue restricts the range of v_1 to $[2, 3]$. The reason is that if v_1 has 4 children, then v_2 could not have at least 2, as together with r that would make the total size of the tree 7, contradicting the clue of r.

Our labeling algorithms require some consistency among the labels of different nodes. However, since insisting that input clues be consistent may be too restrictive, one may think of allowing arbitrary clues, at the expense of giving the labeling algorithm the ability to truncate the incoming clue to the largest consistent subclue possible. In this section we discuss this second approach and its overhead.

For each node v already inserted we define two ranges, each of which is contained in its original subtree clue. The *current subtree range* of v, denoted by $[\ell^s(v), h^s(v)]$,

is defined such that $\ell^s(v)$ is the smallest size of a subtree rooted by v after any legal continuation of the insertion sequence, and $h^s(v)$ is the largest size of such subtree. The *current future range* of v, denoted by $[\ell^f(v), h^f(v)]$, is defined such that $\ell^f(v)$ is the smallest number of descendants of *future* children of v after any legal continuation of the insertion sequence, and $h^f(v)$ is the largest possible number of such nodes. (By future children of v we mean children of v that were not yet inserted.) Note that in contrast with the clue of v, these ranges associated with v are dynamic and narrow down as new nodes are inserted into the tree. It is not hard to see that indeed the set of all possible integers k such that there is a consistent continuation of the sequence, where k is the size of the subtree rooted by v at the end, forms an interval. Similarly, integers k such that there is a continuation, where k is the number of descendants of future children of v at the end, form an interval.

We can now define more precisely when an insertion is consistent. Say we insert a node v, with parent $p(v)$, and a subtree clue $[\ell(v), h(v)]$. Let $[\ell^f(p(v)), h^f(p(v))]$ be the current future range of $p(v)$ before the insertion of v. The subtree clue is consistent if and only if $h(v) \leq h^f(p(v))$.

With sibling clues the definition of consistency is somewhat more involved. We require that each point in the subtree range of the clue and each point in the sibling range of the clue can be realized by some continuation of the sequence without violating any clue of a previously inserted node.

Say we insert a node v, with a sibling clue that consists of a subtree range $[\ell(v), h(v)]$ and a sibling range $[\overline{\ell}(v), \overline{h}(v)]$. Let $[\ell^f(p(v)), h^f(p(v))]$ be the current future range of $p(v)$ before the insertion of v. Clearly we must have that $h(v) \leq h^f(p(v)) - \overline{\ell}(v)$, as otherwise it is not possible to insert both $h(v)$ descendants of v and at least $\overline{\ell}(v)$ descendants of future siblings of v which are also descendants of future children of $p(v)$. Similarly, we must have that $\overline{h}(v) \leq h^f(p(v)) - \ell(v)$. We must also have that $\ell(v) + \overline{h}(v) \geq \ell^f(p(v))$ and that $\overline{\ell}(v) + h(v) \geq \ell^f(p(v))$.

It follows that if we know the current future range of $p(v)$ when v is inserted and the clue of v is inconsistent, then we can truncate it as follows. If the clue is a subtree clue, all we have to do is to set $h(v) := \min\{h(v), h^f(p(v))\}$. If $\ell(v) > h(v)$ after resetting $h(v)$, then there is no way we can truncate the clue to be consistent. Similarly, if the clue is a sibling clue, we set $h(v) = \min\{h(v), h^f(p(v)) - \overline{\ell}(v)\}$, and $\overline{h}(v) = \min\{\overline{h}(v), h^f(p(v)) - \ell(v)\}$. If afterwards $\ell(v) > h(v)$ or $\overline{\ell}(v) > \overline{h}(v)$, there is no way we can truncate the sibling clue to be consistent.

The following lemma suggests how to compute the current subtree range and the current future range of every node for an insertion sequence with subtree clues. The proof is rather straightforward and thus omitted.

LEMMA 4.1. *The lower bound of the current subtree range of every node v can be computed while traversing the tree bottom up using the following recursion:*

$$\ell^s(v) = \max\left\{\ell(v), 1 + \sum_{\{u|p(u)=v\}} \ell^s(u)\right\}. \tag{4.1}$$

The upper bound of v's current subtree range can be recursively computed (top down) as follows.

For the root node r, $h^s(r) = h(r)$. For a node v with parent $p(v)$,

$$h^s(v) = \min\left\{h(v), h^s(p(v)) - 1 - \sum_{\{u|u\neq v \wedge p(u)=p(v)\}} \ell^s(u)\right\}. \tag{4.2}$$

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

$$\ell^f(v) = \max\left\{0, \ell^s(v) - 1 - \sum_{\{u|p(u)=v\}} h^s(u)\right\}, \tag{4.3}$$

$$h^f(v) = h^s(v) - 1 - \sum_{\{u|p(u)=v\}} \ell^s(u)\ . \tag{4.4}$$

This lemma also shows how to update the current subtree range and the current future range of every node as new nodes are inserted. When the root is inserted we have $\ell^s(r) = \ell(r)$, $h^s(r) = h(r)$, $\ell^f(r) = \ell(r) - 1$, and $h^f(r) = h(r) - 1$. Assume we insert a node u with $p(u) = v$, with a consistent subtree clue, that is $h(u) \leq h^f(v)$. Then $\ell^s(u) = \ell(u)$ and $h^s(u) = h(u)$. The current future range of the new node u is $\ell^f(u) = \ell(u) - 1$ and $h^f(u) = h(u) - 1$. The insertion of u may change the lower bound of the current subtree range $\ell^s(v)$ for every ancestor v of u. Siblings of any such ancestor that changes its $\ell^s(v)$ and their descendants may change their upper bounds $h^s(v)$. For each node whose current subtree range changes we recalculate its current future range and the current future range of its parent.

One can also devise a similar, and somewhat more evolved, algorithm to maintain current ranges for sibling clues.

We had seen that precise tracking of current ranges may necessitate updates throughout the tree as a result of a single insertion. Therefore if incoming clues are not guaranteed to be consistent, it may be quite expensive for the labeling algorithm to truncate them.

Fortunately, to establish worst-case bounds on the length of the labels produced by our algorithms it is enough for the clues to be only partially consistent. To define when a sequence is partially consistent we first define an upper bound on $h^f(v)$. For sibling clues we define $h^{rf}(v) = h(v) - (\sum_{u|p(u)=v} \ell(u)) - 1$, where the sum is over all children of v already inserted into the tree. We say that an insertion sequence with subtree clues is partially consistent if when we insert a node w, $h(w) \leq h^{rf}(p(w))$, where $h^{rf}(p(w))$ is calculated before w is inserted into the tree, i.e., without counting w as a child of $p(w)$.

For sibling clues we define the bound $h^{rf}(v)$ to be $\overline{h}(u)$, where u is the last child of v which we inserted, or $h(v) - 1$ if v has no children yet. A sequence of insertions with sibling clues is *partially consistent* if when we insert a node v with subtree range $[\ell(v), h(v)]$ and sibling range $[\overline{\ell}(v), \overline{h}(v)]$, then $h(v) \leq h^{rf}(v) - \overline{\ell}(v)$, and $\overline{h}(v) \leq h^{rf}(v) - \ell(v)$.

If clues arrive with no guarantee on consistency, then it is not hard to truncate them (if possible) to be partially consistent. All we need to do is to maintain for every node v the upper bound $h^{rf}(v)$. When a new child, w, of v is inserted we set $h(w) = \min\{h(w), h^{rf}(v)\}$ and update $h^{rf}(v) := h^{rf}(v) - \ell(w)$. Similarly, if we maintain $h^{rf}(v)$, for every node v, we can truncate the sibling clue that arrived with a node w and then set $h^{rf}(p(w)) = \overline{h}(w)$.

In the rest of this paper we assume that clues are at least partially consistent.

Example. Consider an insertion sequence with *subtree* clues, and let $\rho = 2$. Assume that we first insert a root r with a subtree range of $[5, 10]$. Then we insert a child v of r with a subtree range of $[4, 8]$. At this point the current future range of r is $[\ell^f(r), h^f(r)] = [0, 5]$. Also the bound $h^{rf}(r)$ which in general may be larger than $h^f(r)$ equals 5 in this case. This future range indicates that if case r would have

children other than v, then these children would not have more than 5 descendants altogether. Indeed, if these future children of r have more than 5 descendants, then together with the 4 children that v must have, and r itself, we contradict the upper bound of the subtree range of r. Now assume a child of u of v is inserted with a subtree range of $[5, 7]$; then the current subtree range of v becomes $[6, 8]$, and therefore the future range of r goes down to $[0, 3]$. The upper bound $h^{rf}(r)$, however, remains 5.

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 bounds is at most a factor of two. Thus, the current future range of r after the insertion of v will be contained in one of the following intervals $[0, 0]$: $[1, 2]$, $[2, 4]$, or $[3, 5]$. As soon will become evident, this seemingly small difference is what makes much more efficient labeling 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 length of the labels) . 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 of r after the insertion of v and their descendants. This, together with the label of the root itself, implies that the domain of labels must contain at least 14 labels.

5. Lower bound for subtree clues. We first establish a lower bound on the length of the maximum label of any labeling algorithm with subtree clues.

THEOREM 5.1. *Let A be a labeling algorithm with ρ-tight subtree clues, where $\rho > 1$. There is an infinite family of insertion sequences on which the length of the maximum label assigned by A is $\Omega(\log^2(n))$ bits, where $n = h(r)$ is the upper bound on the subtree clue of the root, and thereby proportional to the length of the sequence.*

Proof. For every n (not necessarily an integer) we define a family of sequences, $\mathcal{F}(n)$, as follows. Let $T = \max\{4\rho, \frac{4\rho}{\rho-1}\}$. If $n \leq T$, then $\mathcal{F}(n)$ consists of a single sequence of a single node with subtree clue $[1, \rho]$.

For $n > T$ we define $\mathcal{F}(n)$ as follows. Let k be an integer such that $\frac{n}{4\rho} < k \leq \frac{n}{2\rho}$. Each sequence in $\mathcal{F}(n)$ starts with k insertions of nodes $v_0, \ldots, v_{k-1}$, where v_0 is the root and v_{i+1} is a child of v_i for every $0 \leq i < k$. For $0 \leq i < k$ the subtree clue of v_i is $[\frac{n}{\rho} - i, n - i\rho]$. See Figure 5.1.

After we insert $v_0, \ldots, v_{k-1}$ the current future range of v_i, for $0 \leq i < k-1$, is $[0, n-i\rho-\lceil\frac{n}{\rho}-(i+1)\rceil-1] \supseteq [0, \frac{\rho-1}{\rho}n-i(\rho-1)-1]$ since, among the $n-i\rho$ descendants that v_i can contain according to its subtree clue, $\lceil\frac{n}{\rho} - (i+1)\rceil$ must be descendants of v_{i+1} and we also have to exclude v_i itself. Clearly, since $k \leq \frac{n}{2\rho}$, we obtain that for every $0 \leq i < k-1$ the current future range of v_i contains the interval $[0, \frac{\rho-1}{\rho}\frac{n}{2} - 1]$. The current future range of v_{k-1} is $[0, n - (k-1)\rho - 1]$, which contains the range $[0, \frac{\rho-1}{\rho}\frac{n}{2} - 1]$. Also note that since $n > \frac{4\rho}{\rho-1}$ we have that $[0, \frac{\rho-1}{\rho}\frac{n}{2} - 1] \supseteq [0, \frac{\rho-1}{\rho}\frac{n}{4}]$.

Now consider the sequences in $\mathcal{F}(\frac{\rho-1}{\rho}\frac{n}{4})$. For each sequence s in $\mathcal{F}(\frac{\rho-1}{\rho}\frac{n}{4})$ and $i \leq k-1$ we obtain a sequence in $\mathcal{F}(n)$ that starts with $v_0, \ldots, v_{k-1}$ and continues as s, where the first vertex in s is inserted as a child of v_i.

Consider a deterministic algorithm A and the labels it assigns to the nodes of the insertion sequences of $\mathcal{F}(n)$. Let s and s' be two insertion sequences in $\mathcal{F}(n)$ with a common prefix p. (Note that every two insertion sequences in $\mathcal{F}(n)$ have a nonempty common prefix.) Since the algorithm is deterministic, insertions in p must get identical labels in s and in s'. We claim, however, that for every insertion $x \in s \setminus p$ and insertion $y \in s' \setminus p$ the label of x must be different from the label of y.

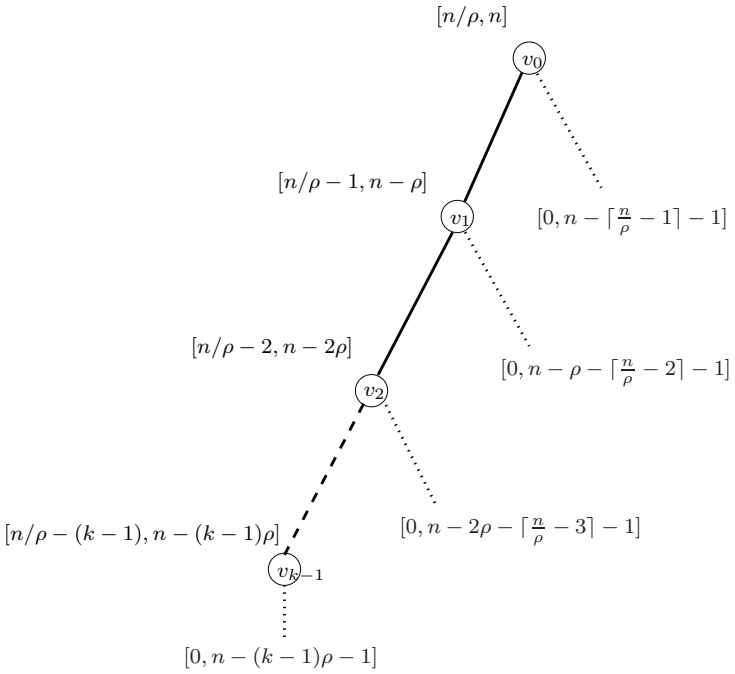

FIG. 5.1. *The beginning of an insertion sequence in $\mathcal{F}(n)$. We insert a path of k node $v_0, \ldots, v_{k-1}$. The subtree clue of each node is depicted to the left of the node. The current future range of a node is depicted at the end of the dashed edge emanating from it.*

To establish this claim note that by the definition of $\mathcal{F}(n)$ there are two different insertions u and u' in p such that u' is a descendant of u, and either (1) x is a descendant of u, y is a descendant of u', and x is not a descendant of u', or, symmetrically, (2) y is a descendant of u, x is a descendant of u', and y is not a descendant of u'. We assume that (1) occurs; if case (2) occurs, the argument is symmetric. If x and y get the same label, then either we report that y is not a descendant of u' in s', or we report that x is a descendant of u' in s. In either case this contradicts the correctness of the labeling algorithm.

It follows from our discussion that a deterministic algorithm assigns a different label to the last node of each sequence of $\mathcal{F}(n)$ so we can establish a lower bound by calculating the cardinality of $\mathcal{F}(n)$. Let $f(n) = |\mathcal{F}(n)|$, and note that $f(n) \geq f(m)$ if $n \geq m$. (This could be easily established by induction.) Then,

$$f(n) \geq \left(\frac{n}{4\rho}\right) f\left(\frac{n}{4}\left(\frac{\rho-1}{\rho}\right)\right)$$
$$\geq \frac{n}{4\rho}\frac{n\frac{\rho-1}{4\rho}}{4\rho}\frac{n(\frac{\rho-1}{4\rho})^2}{4\rho} .$$

So we conclude that

$$f(n) \geq \left(\frac{n}{4\rho}\right)^{\Omega(\log(n))/\log(4\rho/(\rho-1))} .$$

By taking logarithms we obtain the lemma for deterministic algorithms. To obtain the lower bound for randomized algorithms we impose a uniform distribution

on $\mathcal{F}(n)$. By the discussion above the expected maximum label length of any deterministic algorithm on this distribution is $\Omega(\log^2 n)$, so by using Yao's lemma the proof follows. □

For sibling clues we cannot prove a superlogarithmic lower bound on the maximum label length. A logarithmic lower bound follows from the fact that labels should be unique.

Next we define *integer marking* of a tree. Integer marking will come in handy to design labeling algorithms that use the clues.

6. Integer marking and labeling schemes. An integer marking algorithm assigns to each node v when it is inserted into the tree an integer $N(v) \geq 1$ such that, at the end of the insertion sequence, for every node v

$$N(v) \geq 1 + \Sigma_{\{u|v=p(u)\}} N(u) \ . \tag{6.1}$$

Given a labeling algorithm A we can derive from A an integer marking algorithm A_m as follows. Let v be the node which we currently insert, and let $B(v)$ be the set of different labels that A assigns to a descendant of v (not necessarily proper) in any possible continuation of the insertion sequence. The integer which A_m associates with v is $|B(v)|$. Note that the set $B(v)$ is not only a function of the node v and its clue, but it also may depend on the prefix of the insertion sequence (and its associated clues) up to the insertion of v.

Let $u_1, u_2, \ldots, u_k$ be the children of v in the tree obtained at the end of the insertion sequence. Since A is a correct labeling algorithm we obtain that $B(u_i) \cap B(u_j) = \emptyset$. Furthermore, by the definition of the sets $B(w)$ we have that for all $i, B(u_i) \subset B(v)$, and by the correctness of A, the label of v is contained in $B(v)$ but is not contained in any of the $B(u_i)$'s. Therefore the marking of A_m indeed satisfies (6.1) as required.

LEMMA 6.1. *If there is a labeling algorithm that uses at most $f(n)$ bits to label a sequence of length n, then there is an integer marking algorithm that marks the sequence such that $N(r) = 2^{f(n)}$.*

Our main use of integer markings, however, will be the "converse" of Lemma 6.1. That is a way to derive a labeling algorithm from an integer marking algorithm.

Given an integer marking algorithm we first show how to derive an interval labeling algorithm from it. The length of the labels produced by this interval labeling algorithm is at most $2\lceil \log N(r) \rceil$. Then we also show how to derive a prefix labeling algorithm from the integer marking algorithm. The length of the labels produced by the prefix labeling algorithm is at most $\log N(r) + d$, where d is the depth of the final tree.

Interval scheme. For each node v, with assigned range label $[r_1(v), r_2(v)]$ we maintain an integer $s(v)$ such that the subrange $[r_1(v), s(v) - 1]$ is already used by descendants of v, and the subrange $[s(v), r_2(v)]$ is still free for future children of v and their descendants. The root r is labeled by the interval $[1, N(root)]$, and we set $s(r) = 1$. When a node u is inserted and assigned an integer $N(u)$ by the marking algorithm, we label it with the interval $[s(p(v)), s(p(v)) + N(v) - 1]$, we set $s(p(v)) := s(p(v)) + N(v)$, and $s(u) = s(p(v))$. It is easy to check that since the marking algorithm satisfies (6.1) our labeling algorithm indeed produces correct intervals; i.e., the interval of every node u is properly contained in the interval of its parent.

Prefix scheme. The root is labeled by the empty string. When the ith child, u_i, of a node v is inserted, it is labeled by the label of v concatenated with a string

s_i, such that (i) $s_1, \ldots, s_i$ are prefix-free, and (ii) $|s_i| = \lceil \log(N(v)/N(u_i)) \rceil$. It is easy to see that this gives us a prefix labeling such that the length of the longest label is bounded by $\log(N(root)) + d$. It remains to show how to decide on a string s_i when u_i is inserted.

Consider the fractions $f_i = N(u_i)/N(v)$. Note that f_i is known when u_i is inserted. Let $f'_i = 1/2^j$, where j is the minimum such that $f'_i \leq f_i$, i.e., $j = \lceil \log(N(v)/N(u_i)) \rceil$. Let D be the minimum such that $2^D \geq N(v)$. Clearly $f'_i \geq 1/2^D$ for every i, and since N is an integer marking it follows that

$$\sum_i f'_i < 1 \ . \tag{6.2}$$

Multiplying both sides of (6.2) by 2^D, we obtain that if $a_i = f'_i * 2^D$, then $\sum a_i < 2^D$. The following lemma shows that if we are given online a sequence $a_1, \ldots, a_n$ of powers of 2, such that $\sum a_i \leq 2^D$ and $a_i \geq 1$, then we can place each a_i in a node v_i of a complete binary tree with 2^D leaves such that the following hold:

1. Node v_i has a_i leaf descendants.
2. Node v_i is not an ancestor of v_j for every i and j.

We label each edge in T going to a left child by 0 and an edge going to a right child by 1. Once placing a_i in v_i the string s_i is just the concatenation of the labels on the path from the root to v_i. The number of edges on the path from the root to v_i is

$$D - \log a_i = D - (D + \log f'_i) = \lceil \log(N(v)/N(u_i)) \rceil$$

as required.

LEMMA 6.2. *Let $a_i = 2^{x_i}$, $i = 1, \ldots, n$, be a sequence of powers of 2 that arrives online such that $\sum_i a_i \leq 2^D$. Let T be a complete binary tree with 2^D leaves. We can place a_i, when it arrives, in a node $v_i \in T$ such that v_i has a_i leaf descendants and every pair v_i, v_j is unrelated.*

Proof. We define a node v to be *free* if neither of the ancestors of v nor the descendants of v contains an item. We use the following algorithm. When a_i arrives we place it in the leftmost node v such that the following hold:

1. Node v has a_i leaf descendants.
2. Node v is free.

We claim that this algorithm never gets stuck without an appropriate node for a_i. Let $R = 2^D - \sum_{j=1}^{i-1} a_j$. Note that after placing $a_1, \ldots, a_{i-1}$ there are R free leaves in T. Let m be the maximum such that $2^m \leq R$. To prove the claim we show that after placing $a_1, \ldots, a_{i-1}$ there is a free node with at least 2^m leaf descendants. Assume for a contradiction that there is not such a node. Let G be the set of minimal (with respect to depth) free nodes in T. The nodes of G induce a partition of the R free leaves in T where each free leaf is associated with its ancestor in G. The size of each set in the partition is a power of 2. Since the sum of the sizes of the sets is R and there is no set of size greater than 2^m, there must be at least 2 sets of some particular size 2^p, where $p < m$. Let v_1 and v_2 be the nodes in G which correspond to these two sets. Assume without loss of generality that v_1 is to the left of v_2. Consider the sibling w of v_2. Since v_2 is a minimal free node, w is not free and must have a descendant containing an item a_j, $j < i$ (in particular $w \neq v_1$). Since v_1 is free this contradicts our rule of placing an item in the leftmost free node of the appropriate size. ∎

We summarize what we have just proved in the following theorem.

THEOREM 6.3. *Given an integer marking algorithm we can derive from it an interval labeling scheme with maximum label of length $2\lceil \log(N(r)) \rceil$ bits and a prefix*

labeling scheme of maximum label of length $\log(N(r)) + d$ *bits, where* d *is the depth of the resulting tree.*

Consider the simple case when $\rho = 1$; i.e., the subtree clue of every node v in fact gives the size of the subtree rooted by v exactly. In this case if we let $N(v) = \ell(v) = h(v)$, then we obtain an integer marking. By Theorem 6.3 we can convert this integer marking algorithm to a labeling algorithm using either of the techniques described above. With interval labels we obtain a labeling with maximum label length $2(\lceil \log(n) \rceil)$, and with prefix labels we obtain a labeling with maximum label length $\log(n) + d$.

To deal with the more interesting case when $\rho > 1$ we use the notion of a *partial integer marking*. A partial integer marking distinguishes between nodes v such that $h(v) > c(\rho)$ and nodes v such that $h(v) \leq c(\rho)$, where $c(\rho)$ is a fixed constant that depends only on ρ. We call $c(\rho)$ the *threshold* of the partial integer marking. We refer to nodes of the first kind as *heavy nodes* and nodes of the second kind as *light nodes*. Note that consistency of the insertion sequence implies that a child of light node must be a light node. A partial integer marking assigns to each node v, when it is inserted into the tree, an integer $N(v) \geq 1$ such that, at the end of the insertion sequence, for every heavy node v,

$$N(v) \geq 1 + \Sigma_{\{u|v=p(u)\}} N(u) \ , \tag{6.3}$$

and for every light node v, $N(v) = 1$.

Although a partial integer marking relaxes the requirement from the marks of light nodes, we can still derive a labeling algorithm from it with asymptotic label length similar to what we had for integer marking. We obtain this labeling as follows. When a heavy node, or a light node whose parent is heavy, is inserted, we label it with an interval label or a prefix label as defined above for integer marking. Since heavy nodes satisfy (6.3) this labeling is well defined. It covers all heavy nodes and their children. We refer to these labels as the *top labels*. We still have to define the labels of light nodes with light parents.

Consider a light node v such that $p(v)$ is heavy, and let $T(v)$ be the subtree of light nodes rooted by v. We label each node in such subtree $T(v)$ (when it arrives) using a simple prefix scheme from section 3 that completely ignores the clues. Let u be a light node whose parent is also a light node. Let $a(u)$ be the light ancestor of u of minimal depth. We label u with a pair. The first component of this pair is the top label of $a(u)$. The second component of the pair is a label of u according to the simple labeling of $T(a(u))$. It is easy to see that we can assign these labels online.

Notice that the collection of nodes labeled by top labels forms only a subtree. The nodes labeled by pairs are partitioned into a collection of disjoint trees[3] each rooted by a light node whose parent is heavy.

Next we describe the algorithm to answer ancestor queries based on these labels. The proof of its correctness is straightforward. Let u and v be two nodes such that we have to decide whether u is a descendant of v. There are four cases.

1. The label of u and the label of v both consist only of top labels. In this case we answer using the query algorithm for the top labels.
2. The label of u consists only of a top label, and the label of v is a pair. In this case u cannot be a descendant of v.
3. The label of v consists only of a top label, and the label of u is a pair. We extract the top label part of the label of u. This is the label of the light

[3]The root of each of these trees is not labeled by a pair; only the internal nodes are labeled.

ancestor of u of minimal depth, denoted by $a(u)$. We use the top label of $a(u)$ and the top label of v to determine if $a(u)$ is a descendant of v using the query algorithm for top labels. In this case node u is a descendant of v if and only if $a(u)$ is a descendant of v.

4. The label of v and the label of u are pairs. In this case for u to be a descendant of v the following two conditions have to hold:
(1) The top label in the pair of u should be identical to the top label in the pair of v, which implies that the light ancestor of v of minimal depth is the same as the light ancestor of u of minimal depth. Denote this ancestor by w.
(2) Node u is a descendant of v according to the labeling of $T(w)$. We determine whether this condition holds using the query algorithm of the labeling of $T(w)$. The labels of u and v in this labeling are the second components of the corresponding pairs.

Clearly the length of the top labels is bounded by $2\lceil \log N(r) \rceil$ if we use interval labels and by $\log N(r) + d$ if we use prefix labels by the same argument as for integer marking. Let v be a light node whose parent is heavy. Since $h(v) \leq c(\rho)$, $T(v)$ contains at most $c(\rho)$ nodes, so the length of a label of a node in the labeling of $T(v)$ is $O(c(\rho))$ bits. Summing up we obtain that the total length of a label is $2\lceil \log N(r) \rceil + c(\rho)$ bits if we use interval labels and $\log N(r) + d + c(\rho)$ bits if we use prefix labels. We conclude with the following theorem.

THEOREM 6.4. *Given a partial integer marking algorithm with threshold $c(\rho)$ we can derive from it an interval labeling scheme with maximum label of length $2\lceil \log(N(r)) \rceil + c(\rho)$ bits and a prefix labeling scheme of maximum label of length $\log(N(r)) + d + c(\rho)$ bits, where d is the depth of the resulting tree.*

7. Labeling with subtree clues. We now develop a partial integer marking for a sequence with subtree clues for some $\rho > 1$. Let

$$f(n) = n^{1+\log_{\frac{\rho}{\rho-1}} n} .$$

We prove the following theorem.

THEOREM 7.1. *Consider a sequence with consistent subtree clues. Let $c(\rho) = \max\{\frac{\rho^2}{\rho-1}, 8\rho \ln(8\rho)\}$ be the threshold for heavy nodes. Then the marking $N(u) = \lfloor f(h(u)-1) \rfloor + 1$ for heavy node u is a partial integer marking.*

We start by observing some basic properties of $f(n)$. Since $\log_{\frac{\rho}{\rho-1}} n$ is nonnegative for $n \geq 1$, $f(n)$ grows faster than a linear function. So for $n \geq 1$,

$$f(n+1) \geq f(n) + 1 . \tag{7.1}$$

Another property of f that we use is specified in the following lemma.

LEMMA 7.2. *For $n \geq \max\{\frac{\rho^2}{\rho-1}, 8\rho \ln(8\rho)\}$, and for every $\rho + 1 \leq k \leq n$, $f(n) \geq 1 + f(k-1) + f(n - \frac{k}{\rho})$.*

Proof. Consider first the function $g(x) = f(x-1) + f(n - \frac{x}{\rho})$ for $x \geq 1$. Its derivatives are $g'(x) = f'(x-1) - \frac{1}{\rho} f'(n - \frac{x}{\rho})$ and $g''(x) = f''(x-1) + \frac{1}{\rho^2} f''(n - \frac{x}{\rho})$. Writing $f(x) = e^{\ln(x)(1+\log_{\frac{\rho}{\rho-1}}(x))}$ we see that

$$f'(x) = e^{\ln(x)(1+\log_{\frac{\rho}{\rho-1}}(x))} \left(\frac{1}{x} + \frac{2 \log_{\frac{\rho}{\rho-1}}(x)}{x} \right)$$

and

$$f''(x) = e^{\ln(x)(1+\log_{\frac{\rho}{\rho-1}}(x))}\left(\frac{2\log_{\frac{\rho}{\rho-1}}(x)}{x^2} + \frac{4\log^2_{\frac{\rho}{\rho-1}}(x)}{x^2} + \frac{2}{\ln(\rho/(\rho-1))x^2}\right) .$$

Clearly $f''(x) \geq 0$ for $x \geq 1$, and therefore since $n \geq c(\rho) > \frac{\rho}{\rho-1}$ (so $n - \frac{n}{\rho} \geq 1$) $g''(x) \geq 0$ for $x \in [1, n]$. This implies in particular that $g(x)$ is convex in the interval $[\rho+1, n]$, and therefore $g(n)$ or $g(\rho+1)$ is larger than $g(x)$ for every $x \in [\rho+1, n]$. Now,

$$g(n) = (n-1)^{1+\log_{\frac{\rho}{\rho-1}}(n-1)} + \left(n - \frac{n}{\rho}\right)^{1+\log_{\frac{\rho}{\rho-1}}\left(n-\frac{n}{\rho}\right)}$$

and

$$g(\rho+1) = \rho^{1+\log_{\frac{\rho}{\rho-1}}(\rho)} + \left(n - \frac{\rho+1}{\rho}\right)^{1+\log_{\frac{\rho}{\rho-1}}\left(n-\frac{\rho+1}{\rho}\right)} .$$

Since $n - \frac{\rho+1}{\rho} \leq n-1$, and $\rho \leq n - \frac{n}{\rho}$ for $n \geq \frac{\rho^2}{\rho-1}$, and $f(x)$ is monotone, we see that $g(n) \geq g(\rho+1)$. So to finish the proof we have to show that $1 + g(n) \leq f(n)$ or alternatively that $\frac{1+g(n)}{f(n)} \leq 1$. Substituting for $g(n)$ we obtain that

$$1 + g(n) = 1 + (n-1)^{1+\log_{\frac{\rho}{\rho-1}}(n-1)} + \left(n\frac{\rho-1}{\rho}\right)^{1+\log_{\frac{\rho}{\rho-1}}\left(n\frac{\rho-1}{\rho}\right)} . \tag{7.2}$$

By dividing both sides by $f(n) = n^{1+\log_{\frac{\rho}{\rho-1}} n}$ we obtain that

$$\frac{1+g(n)}{n^{1+\log_{\frac{\rho}{\rho-1}} n}} \leq \frac{1}{n^{1+\log_{\frac{\rho}{\rho-1}} n}} + \left(1 - \frac{1}{n}\right)^{1+\log_{\frac{\rho}{\rho-1}}(n-1)} + \frac{1}{n^2} .$$

We recall the inequality $\left(1 - \frac{1}{n}\right)^x \leq 1 - \frac{x}{2n}$ which holds for $x \leq n/2$, and using it for $x = 1 + \log_{\frac{\rho}{\rho-1}}(n-1)$ we obtain that

$$\frac{1+g(n)}{n^{1+\log_{\frac{\rho}{\rho-1}} n}} \leq \frac{1}{n^{1+\log_{\frac{\rho}{\rho-1}} n}} + 1 - \frac{1+\log_{\frac{\rho}{\rho-1}}(n-1)}{2n} + \frac{1}{n^2} .$$

Note that indeed $x = 1 + \log_{\frac{\rho}{\rho-1}}(n-1) \leq n/2$ for $n \geq c(\rho)$.[4]

Since $n \geq \frac{\rho}{\rho-1}$ the first term on the right-hand side is not larger than $1/n^2$ so that

$$\frac{1+g(n)}{n^{1+\log_{\frac{\rho}{\rho-1}} n}} \leq 1 - \frac{1}{2n} + \frac{2}{n^2} ,$$

and since the right-hand side is smaller than 1 for $n \geq 4$, we obtain that $1+g(n) \leq f(n)$ for $n \geq c(\rho)$, and the lemma follows. □

[4]To see this note that $\log_{\frac{\rho}{\rho-1}}(n-1) = \frac{\ln(n-1)}{\ln(1+\frac{1}{\rho-1})} \leq \rho\ln(n-1)$, where the last inequality holds since $\frac{x}{1+x} \leq \ln(1+x)$ for all $x > -1$. So since for $n > c(\rho)$, $n/4 < n/2 - 1$, it suffices to show that $\rho\ln(n) \leq n/4$, which is true for $n \geq 8\rho\ln(8\rho)$.

From (7.1) we obtain that

$$\lfloor f(n+1)\rfloor \geq \lfloor f(n)\rfloor + 1, \tag{7.3}$$

and from Lemma 7.2 we obtain that for $n \geq c(\rho)$, and for every $\rho + 1 \leq k \leq n$,

$$\lfloor f(n)\rfloor \geq 1 + \lfloor f(k-1)\rfloor + \left\lfloor f\left(n - \frac{k}{\rho}\right)\right\rfloor \ . \tag{7.4}$$

We now prove Theorem 7.1, by showing that at any time

$$N(v) \geq 1 + \sum_{\{u|p(u)=v\}} N(u) + \lfloor f(h^{rf}(v))\rfloor \tag{7.5}$$

for every heavy node v. The proof is by induction on the insertion sequence. When we first insert v, $N(v) = \lfloor f(h(v)-1)\rfloor + 1$, and $h^{rf}(v) = h(v) - 1$, and v has no children, so (7.5) holds for v. Assume now that (7.5) holds before we insert a new child w of v. We show that (7.5) continues to hold after this insertion. We denote by h the value of $h^{rf}(v)$ before the insertion of w and by h' the value of $h^{rf}(v)$ after the insertion. There are two cases.

Node w is light. In this case $N(w) = 1$ and $h' + 1 \leq h$. So from (7.3) it follows that $\lfloor f(h)\rfloor \geq \lfloor f(h')\rfloor + 1$. So after the insertion $\sum_{\{u|p(u)=v\}} N(u)$ increases by 1 but $\lfloor f(h^{rf}(v))\rfloor$ decreases by at least 1 so that (7.5) holds after the insertion.

Node w is heavy. In this case by definition $N(w) = \lfloor f(h(w)-1)\rfloor + 1$. Therefore to show that (7.5) holds after the insertion of w it suffices to show that

$$\lfloor f(h(w)-1)\rfloor + 1 + \lfloor f(h')\rfloor \leq \lfloor f(h)\rfloor \ . \tag{7.6}$$

Since $h' \leq h - h(w)/\rho$, (7.6) holds if

$$\lfloor f(h(w)-1)\rfloor + 1 + \lfloor f(h - h(w)/\rho)\rfloor \leq \lfloor f(h)\rfloor, \tag{7.7}$$

which follows from (7.4) and since w is heavy.

This concludes the proof of Theorem 7.1. Combining this theorem with Theorem 6.4 we obtain the following theorem.

THEOREM 7.3. *There is a labeling algorithm for sequences with subtree clues that uses labels of length $O(\log^2(n))$ bits.*

8. Labeling with sibling clues. We now develop a partial integer marking for sequences with sibling clues. Let

$$f(n) = n^{2/\log(1+\frac{1}{\rho})} \ .$$

We prove the following theorem.

THEOREM 8.1. *Consider a sequence with consistent sibling clues. The integer marking $N(u) = \lfloor f(h(u)-1)\rfloor + 1$ is a partial integer marking with threshold $c(\rho) = 2\rho^2$.*

First we observe some basic properties of $f(n)$. Since $\rho > 1$, we get that $\log(1 + \frac{1}{\rho}) \leq 1$ and $f(n)$ in fact grows faster than a quadratic function so that, in particular for $n \geq 1$,

$$f(n+1) \geq f(n) + 1 \ . \tag{8.1}$$

Another property of f that we use is specified in the following lemma.

LEMMA 8.2. *Let $c(\rho) = 2\rho^2$. For $n \geq c(\rho)$ and for every $\rho \leq k \leq \rho n/(\rho+1)$, $f(n) \geq 1 + f(k-1) + f(n - \frac{k}{\rho})$ and $f(n) \geq 1 + f(k) + f(n - \frac{k}{\rho} - 1)$.*

Proof. Consider the function $g_1(x) = f(x-1) + f(n - \frac{x}{\rho})$ for $x \geq 1$. Its derivatives are $g_1'(x) = f'(x-1) - \frac{1}{\rho} f'(n - \frac{x}{\rho})$ and $g_1''(x) = f''(x-1) + \frac{1}{\rho^2} f''(n - \frac{x}{\rho})$. It is easy to see that $f''(x) \geq 0$ for $x \geq 0$, and therefore $g_1''(x) \geq 0$ for $x \in [1, \rho n]$. In particular $g_1''(x)$ is convex in the interval $[\rho, \rho n/(\rho+1)]$ and obtains its maximum value in this interval at one of its endpoints.

Let $g_2(x) = f(x) + f(n - \frac{x}{\rho} - 1)$. We can similarly show that $g_2''(x) \geq 0$ for $x \in [0, \rho(n-1)]$. Since $\rho n/(\rho+1) \leq \rho(n-1)$ for $n \geq (\rho+1)/\rho$ (and clearly for $n \geq 2\rho^2 \geq (\rho+1)/\rho$) it follows that for $n > c(\rho)$, $g_2''(x)$ is convex in the interval $[\rho, \rho n/(\rho+1)]$ and therefore obtains its maximum value in this interval at one of its endpoints.

So to complete the proof we have to show that

$$f(n) \geq 1 + g_1(\rho) = 1 + f(\rho - 1) + f\,(n-1)\ , \tag{8.2}$$

$$f(n) \geq 1 + g_2(\rho) = 1 + f(\rho) + f\,(n-2)\ , \tag{8.3}$$

$$f(n) \geq 1 + g_1\left(\frac{\rho n}{\rho+1}\right) = 1 + f\left(\frac{\rho n}{\rho+1} - 1\right) + f\left(n - \frac{n}{\rho+1}\right) \tag{8.4}$$

and that

$$f(n) \geq 1 + g_2\left(\frac{\rho n}{\rho+1}\right) = 1 + f\left(\frac{\rho n}{\rho+1}\right) + f\left(n - \frac{n}{\rho+1} - 1\right)\ . \tag{8.5}$$

To establish (8.2) and (8.3) we show that

$$f(n) \geq 1 + f(\rho) + f\,(n-1)\ . \tag{8.6}$$

Using the definition of $f(n)$, (8.6) is equivalent to

$$n^{2/\log(1+\frac{1}{\rho})} \geq 1 + \rho^{2/\log(1+\frac{1}{\rho})} + (n-1)^{2/\log(1+\frac{1}{\rho})}\ .$$

Dividing both sides by $\rho^{2/\log(1+\frac{1}{\rho})}$ we obtain that

$$\left(\frac{n}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} \geq 1/\rho^{2/\log(1+\frac{1}{\rho})} + 1 + \left(\frac{n}{\rho} - \frac{1}{\rho}\right)^{2/\log(1+\frac{1}{\rho})}\ . \tag{8.7}$$

To establish (8.7) it suffices to show that

$$\left(\frac{n}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} - \left(\frac{n}{\rho} - \frac{1}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} \geq 2\ . \tag{8.8}$$

Now,

$$\begin{aligned}
\left(\frac{n}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} - \left(\frac{n}{\rho} - \frac{1}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} &= \left(\frac{n}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} \left(1 - \left(1 - \frac{1}{n}\right)^{2/\log(1+\frac{1}{\rho})}\right) \\
&\geq \left(\frac{n}{\rho}\right)^{2/\log(1+\frac{1}{\rho})} \frac{1}{n} \\
&\geq \frac{n}{\rho^2}\ ,
\end{aligned}$$

which is larger than 2 for $n \geq 2\rho^2$.

To establish (8.4) and (8.5) we show that

$$f(n) \geq 1 + 2f\left(\frac{\rho n}{\rho+1}\right) . \tag{8.9}$$

An easy calculation shows that $f(\frac{\rho}{\rho+1}n) = f(n)/4$. So $1+2f(\frac{\rho}{\rho+1}n) = 1+f(n)/2$, which is smaller than $f(n)$ since for $n \geq 2$, $f(n)/2 \geq 1$. So (8.9) follows. □

From (8.1) we obtain that

$$\lfloor f(n+1) \rfloor \geq \lfloor f(n) \rfloor + 1, \tag{8.10}$$

and from Lemma 8.2 we obtain that for $n \geq 2\rho^2$, and for every $\rho \leq k \leq \frac{n\rho}{\rho+1}$,

$$\lfloor f(n) \rfloor \geq 1 + \lfloor f(k-1) \rfloor + \left\lfloor f\left(n - \frac{k}{\rho}\right)\right\rfloor \tag{8.11}$$

and

$$\lfloor f(n) \rfloor \geq 1 + \lfloor f(k) \rfloor + \left\lfloor f\left(n - \frac{k}{\rho} - 1\right)\right\rfloor . \tag{8.12}$$

We now prove Theorem 8.1 by proving that at any time

$$N(v) \geq 1 + \sum_{\{u|p(u)=v\}} N(u) + \lfloor f(h^{rf}(v)) \rfloor \tag{8.13}$$

for every heavy node v. The proof is by induction on the insertion sequence. When we first insert v, $N(v) = \lfloor f(h(v)-1) \rfloor + 1$, and $h^{rf}(v) = h(v) - 1$ so that (8.13) holds for v that initially has no children. Assume now that (8.13) holds before we insert a new child w of v. We show that (8.13) continues to hold after this insertion. We denote by h the value of $h^{rf}(v)$ before the insertion of w and by h' the value of $h^{rf}(v)$ after the insertion. There are two cases. Note that with consistent sibling clues $h' = \overline{h}(w)$.

Node w is light. The argument for this case is the same as the argument for the same case in section 7.

Node w is heavy. In this case we have to prove that

$$N(w) + \lfloor f(h') \rfloor \leq \lfloor f(h) \rfloor. \tag{8.14}$$

Substituting the definition of $N(w)$ and h' we obtain that (8.14) is equivalent to

$$1 + \lfloor f(h(w)-1) \rfloor + \lfloor f(\overline{h}(w)) \rfloor \leq \lfloor f(h) \rfloor. \tag{8.15}$$

From the consistency of the clues it follows that $\ell(w) \geq h(w)/\rho$, and $\overline{h}(w) \leq h - \ell(w) \leq h - h(w)/\rho$. So (8.15) holds if

$$1 + \lfloor f(h(w)-1) \rfloor + \lfloor f(h - h(w)/\rho) \rfloor \leq \lfloor f(h) \rfloor . \tag{8.16}$$

If $h(w) \leq \frac{\rho n}{\rho+1}$, then (8.16) follows from (8.11) since w is heavy.

From the consistency of the clues it also follows that $\overline{\ell}(w) \geq \overline{h}(w)/\rho$, and $h(w) \leq h - \overline{\ell}(w) \leq h - \overline{h}(w)/\rho$. So (8.15) holds if

$$1 + \lfloor f(h - \overline{h}(w)/\rho - 1) \rfloor + \lfloor f(\overline{h}(w)) \rfloor \leq \lfloor f(h) \rfloor . \tag{8.17}$$

If $h(w) > \frac{\rho n}{\rho+1}$, then $\ell(w) > \frac{n}{\rho+1}$, and by the consistency of the clues $\overline{h}(w) \leq n - \frac{n}{\rho+1} = \frac{\rho n}{\rho+1}$. So (8.17) follows from (8.12) since w is heavy.

This concludes the proof of Theorem 8.1. Combining Theorem 8.1 with Theorem 6.4 we obtain the following theorem.

THEOREM 8.3. *There is a labeling algorithm for sequences with subtree clues that uses labels of length $O(\log n)$ bits.*

9. Bounded-depth trees. We now consider labeling with subtree clues when we have an upper bound of d on the depth of the final tree. From d we can derive an upper bound $d(v)$ on the height of node v by subtracting one from the upper bound on the depth of the parent of v.

We may assume that $h(v) \geq d(v)$; otherwise we can reduce $d(v)$ and set it to be equal to $h(v)$. Furthermore, we modify the definition of a *heavy node* in a partial integer marking (see section 6) as follows. A node is *heavy* if $d(v) \geq c(\rho)$, which clearly implies that $h(v) \geq c(\rho)$. We obtain a labeling from a partial integer marking as in section 6, and Theorem 6.4 still holds.

THEOREM 9.1. *If an upper bound d on the depth of the tree is known in advance, then there is a labeling algorithm for sequences with subtree clues that uses labels of length $O(\log n \log d)$ bits.*

So when d is much smaller than n we gain over the bound on the length of the labels in section 7. Our proof follows along the same footsteps of the proof of Theorem 7.1. Let

$$f(n,d) = n(2(d+1))^{\log_{\frac{\rho}{\rho-1}} n} = n^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))} .$$

Theorem 9.1 follows from the following theorem and Theorem 6.4.

THEOREM 9.2. *Consider a sequence with consistent subtree clues. There is a threshold $c(\rho)$ for which $N(u) = \lfloor f(h(u)-1, d(u))\rfloor + 1$ is a partial integer marking.*

As in section 7, it is easy to check that, for $n \geq 1$,

$$f(n+1,d) \geq f(n,d) + 1 . \tag{9.1}$$

The following lemma is analogous to Lemma 7.2.

LEMMA 9.3. *There exists a threshold $c(\rho)$ such that for $n \geq c(\rho)$, and for every $\rho+1 \leq k \leq n$, $f(n,d) \geq 1 + f(k-1,d-1) + f(n-\frac{k}{\rho},d)$.*

Proof. Consider first the function $g(x) = f(x-1,d-1) + f(n-\frac{x}{\rho},d)$ for $x \geq 1$. Its derivatives with respect to x are $g'(x) = f'(x-1,d-1) - \frac{1}{\rho}f'(n-\frac{x}{\rho},d)$ and $g''(x) = f''(x-1,d-1) + \frac{1}{\rho^2}f''(n-\frac{x}{\rho},d)$. It is easy to see that $f''(x) \geq 0$ for $x \geq 0$, and therefore $g''(x) \geq 0$ for $x \in [1,\rho n]$. This implies in particular that $g(x)$ is convex in the interval $[\rho+1,n]$ (we assume that $c(\rho) \geq \rho+1$ so that $n \geq \rho+1$), and therefore $g(n)$ or $g(\rho+1)$ is larger than $g(x)$ for every $x \in [\rho+1,n]$. Now

$$\begin{aligned} g(n) &= f(n-1,d-1) + f\left(\frac{\rho-1}{\rho}n, d\right) \\ &= (n-1)^{1+\log_{\frac{\rho}{\rho-1}}(2d)} + \left(\frac{\rho-1}{\rho}n\right)^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))} \end{aligned}$$

and

$$\begin{aligned} g(\rho+1) &= f(\rho,d-1) + f\left(n-\frac{\rho+1}{\rho}, d\right) \\ &= \rho^{1+\log_{\frac{\rho}{\rho-1}}(2d)} + \left(n-\frac{\rho+1}{\rho}\right)^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))} . \end{aligned}$$

To prove that $1+g(\rho+1) \le f(n,d)$ we have to show that

$$1+\rho^{1+\log_{\frac{\rho}{\rho-1}}(2d)}+\left(n-\frac{\rho+1}{\rho}\right)^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))} \le n^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))}. \tag{9.2}$$

Equation (9.2) is equivalent to

$$1+\rho^{1+\log_{\frac{\rho}{\rho-1}}(2d)} \le n^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))}-\left(n-\frac{\rho+1}{\rho}\right)^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))}.$$

The right-hand side is larger than

$$n^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))}-(n-1)^{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))},$$

which, by using the binomial expansion of the second term, is larger than

$$\frac{1+\log_{\frac{\rho}{\rho-1}}(2(d+1))}{2}n^{\log_{\frac{\rho}{\rho-1}}(2(d+1))}.$$

Reducing this further by replacing $d+1$ by d, we obtain that to establish (9.2) it is sufficient to show that

$$1+\rho^{1+\log_{\frac{\rho}{\rho-1}}(2d)} \le \frac{1+\log_{\frac{\rho}{\rho-1}}(2d)}{2}n^{\log_{\frac{\rho}{\rho-1}}(2d)}.$$

It is easy to verify that this inequality holds if we pick $c(\rho)=\max\{\rho^2+1,\frac{\rho}{\rho-1}\}$ so that in particular $2d \ge \frac{\rho}{\rho-1}$ and $n \ge \rho^2+1$.

To prove that $1+g(n) \le f(n,d)$ we have to show that

$$1+f(n-1,d-1)+f\left(\frac{(\rho-1)n}{\rho},d\right) \le f(n,d). \tag{9.3}$$

To show this notice that

$$f\left(\frac{(\rho-1)n}{\rho},d\right)=\frac{\rho-1}{\rho}\frac{1}{2(d+1)}f(n,d).$$

Substituting $f(n,d)=2(d+1)\frac{\rho}{\rho-1}f(\frac{(\rho-1)n}{\rho},d)$ and $f(n-1,d-1)=2d\frac{\rho}{\rho-1}f(\frac{(\rho-1)(n-1)}{\rho},d-1)$ in (9.3) we obtain

$$\begin{aligned}1+2d\frac{\rho}{\rho-1}f\left(\frac{(\rho-1)(n-1)}{\rho},d-1\right)+f\left(\frac{(\rho-1)n}{\rho},d\right)\\ \le 2(d+1)\frac{\rho}{\rho-1}f\left(\frac{(\rho-1)n}{\rho},d\right).\end{aligned} \tag{9.4}$$

Equation (9.4) holds since $f(\frac{(\rho-1)n}{\rho},d) \ge 1$ for $n \ge \frac{\rho}{\rho-1}$. □

As in section 7 from (9.1) we obtain that

$$\lfloor f(n+1,d) \rfloor \ge \lfloor f(n,d) \rfloor + 1, \tag{9.5}$$

and from Lemma 9.3 we obtain that for $n \ge c(\rho)=\max\{\rho^2+1,\frac{\rho}{\rho-1}\}$, and every $\rho+1 \le k \le n$,

$$\lfloor f(n,d) \rfloor \ge 1+\lfloor f(k-1,d-1) \rfloor + \left\lfloor f\left(n-\frac{k}{\rho},d\right)\right\rfloor. \tag{9.6}$$

The rest of the proof of Theorem 9.2 is by induction on the insertion sequence as in the proof of Theorem 7.1.

10. Coping with wrong estimates. In this section we explain how the interval and prefix labeling schemes presented in the previous sections can be extended to cope with inconsistent estimates. As we discussed in section 4.1 we can truncate the clues online to make them at least partially consistent. However, this is not always possible, as ranges may become empty. Furthermore, by truncating new clues we implicitly prefer earlier estimates, whereas it may be that the more recent clues are correct but the older ones were wrong.

How to deal with inconsistent clues may be application dependent. The point which we make in this section is that our labeling schemes are flexible enough so that when inconsistency is detected we can still keep labeling and do not have to immediately relabel already labeled nodes. We loosen the guarantees on the length of the labels, but the labeling remains correct. Clearly, the problem is when ranges are underestimated.

Range labeling and inconsistent clues. Recall the interval labeling algorithm from section 6. It uses an integer marking derived from the clue. A node v with mark $N(v)$ is assigned an interval $[r_1(v), r_2(v)]$ of length $N(v)$ within the range of $p(v)$. Since $\sum_{u|v=p(u)} N(u) < N(v)$, as long as labels are consistent, $r_2(v)$ is not used as the right-hand side of an interval assigned to a child of v.

In case of incorrect clues we may have to insert a child u of v when there is no more space within the range of v for the range of u. In this case we use range labels of the form $[r_2(v) \cdot x_1, r_2(v) \cdot x_2]$, where $x_1 \leq x_2$. That is, we add low order bits to the binary encoding of $r_2(v)$ and use them to define a range for u. All endpoints of labels of future descendants of u will start with $r_2(v)$. We can use any labeling scheme to derive these low order bits. In particular we can derive these bits by restarting the same labeling scheme based on integer marking as if u is the root. If later more children of v arrive and the newly allocated bits do not suffice, we further add more low order bits.

We do not have to separate between these new bits that encode x_1 and x_2 from the bits encoding $r_2(v)$. For the decoding to work properly when we do not separate the bits, we think of the left endpoint of each interval as padded with an infinite sequence of 0's and the right endpoint of each interval as padded with an infinite sequence of 1's. We compare endpoints by the lexicographic order of binary strings.

Prefix labeling and inconsistent clues. Recall the prefix labeling scheme of section 6. We use a prefix-free set of strings, corresponding to paths in a complete binary tree T of depth D, to label the children of each node v. Each child u_i of v is associated with a power of two denoted by a_i based on its mark. Then we find a subtree of T with a_i leaves which are unassigned. The path s_i to the root of this subtree is the label we give to u_i.

We proved in Lemma 6.2 that if $\sum_i a_i \leq D$, we can always find an empty subtree of the appropriate size. Furthermore, in our case, as long as clues are correct we even have that $\sum_i a_i < D$. So there is always a string s that together with the strings s_i already assigned to children of v forms a prefix-free collection of strings.

If the clues are incorrect, then when a child u of v is inserted we may not have a subtree of T of the appropriate size. In such a case we use the remaining string s to label u and further descendants of v, as in one of the simple prefix labeling schemes of section 3. Within the subtree rooted by u we can restart a labeling scheme based on integer marking where we concatenate each label produced by this scheme to s.

It should be noted that the more inconsistent the estimates get, 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.

11. Conclusion and related work. One can design labeling schemes in a static or a dynamic setting. The static setting, where the full structure of the tree is known in advance, has been the focus of several previous works [3, 4, 27, 15, 10]. 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.

The problem of designing a persistent labeling scheme for identifying nodes in a sequence of versions of an XML file has been studied by Marian et al. [13], 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. [13] 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 Table 11.1.

TABLE 11.1
Summary of our bounds on the bit length of the labels.

Problem	Static	Dynamic		
		no clues	subtree clues	sibling clues
Bounds	$\Theta(\log n)$	$\Theta(n)$	$\Theta(\log^2 n)$	$\Theta(\log n)$

Often we would like to be able to test whether node u is the parent of node v in the tree T rather than just an ancestor. One could do that (with our labels as well) by attaching the depth of v to the label of v in the tree. Then a node v is a parent of a node u if and only if the labels of v and u indicate that v is an ancestor of u, and the depth of v is one smaller than the depth of u. (For more on labeling for parent queries see [5, 9, 10].)

Following our work, several alternative proposals of dynamic labeling schemes for XML trees have been published. The goal of these schemes was to encode in the labels information about some linear order of the tree in addition to ancestor relationships. (This order is typically an inorder, or the order of the nodes in the file which stores the tree.) Using these schemes one can, from the labels of nodes v and u, determine whether v precedes u in, say, an inorder traversal of the tree, in addition to whether v is an ancestor of u. Naturally, the lower bounds we presented here apply for these more general labels.

We briefly review some of these labeling schemes. The integer-based prefix labeling scheme [17], called the Dewey labeling scheme, assigns the integer i to the ith child of a node. The label of a node is the string obtained by concatenating the integers assigned to all its ancestors. A delimiter is used to separate the integers while concatenating to remove ambiguities. Dewey labels contain much more information than ancestor and parent relations. For example we can determine whether two nodes are siblings, and we can determine the label of the lowest common ancestor of two nodes.

Furthermore, the lexicographic order of these labels is consistent with the order of the nodes in the file. If we want to maintain Dewey labels such that the ith child is labeled i (note that this is not necessary for ancestor relations), then an insertion of a new node may require extensive relabeling. Several papers suggested dynamic schemes related to Dewey that try to alleviate the penalty of relabeling. One such scheme is ORDPATH [14], and a more recent one is DDE [24].

Wu, Lee, and Hsu [22] propose a prefix labeling scheme based on prime numbers. In this labeling scheme, each node is assigned a prime number and the label of a node is the product of all prime numbers assigned to its ancestors. Checking for an ancestor-descendant relationship between two nodes amounts to determining if the label of the descendant is divisible by the label of the ancestor.

Li and Ling [11] and Li, Ling, and Hu [12] show how to assign a codeword to each item in a linear order such that the following hold:

1. The lexicographic order between codewords is consistent with the linear order. That is, if v precedes u, then the codeword of v is lexicographically smaller than the codeword of u.
2. We can add an item anywhere in the order and assign a codeword to it without changing the codewords of other items.

They use a quaternary alphabet in [11] and a binary alphabet in [12]. A central feature of these encodings is that the lexicographic order (rather than numerical order) of codewords corresponds to the linear order of the items. This allows the insertion of a codeword between two existing codewords by using a longer codeword. For example, the codeword 1101 can be inserted between two codewords 11 and 111 without disturbing other existing codewords.

Li and Ling [11] and Li, Ling, and Hu [12] applied such an encoding scheme to the order sensitive components of a labeling scheme to make it dynamic. For example one can use such an encoding to label the leaves when using the simple interval scheme (see section 1). This allows one to add a leaf without any change in existing labels.

Complementary to these works, Silberstein et al. [16] present two I/O-efficient data structures to maintain labels of large and dynamic XML documents. The two data structures, W-BOX (Weight-balanced B-tree for Ordering XML) and B-BOX (Back-linked B-tree for ordering XML), are B-tree based data structures which organize the labels for efficient updates. Their techniques are general and can be incorporated into any labeling scheme.

REFERENCES

[1] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, and T. Rauhe, *Compact labeling scheme for ancestor queries*, SIAM J. Comput., 35 (2006), pp. 1295–1309.

[2] S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann, San Francisco, 1999.

[3] S. Abiteboul, H. Kaplan, and T. Milo, *Compact labeling schemes for ancestor queries*, in Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2001, pp. 547–556.

[4] S. Alstrup and T. Rauhe, *Improved labeling scheme for ancestor queries*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 947–953.

[5] S. Alstrup and T. Rauhe, *Small induced-universal graphs and compact implicit graph representations*, in Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Washington, DC, 2002, pp. 53–62.

[6] D. Barbosa, L. Mignet, and P. Veltri, *Studying the XML web: Gathering statistics from an XML sample*, World Wide Web, 8 (2005), pp. 413–438.

[7] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu, *XQuery 1.0: An XML Query Language*, W3C working draft, http://www.w3.org/TR/xquery (2001).
[8] E. Cohen, H. Kaplan, and T. Milo, *Labeling dynamic XML trees*, in Proceedings of the Symposium on Principles of Database Systems (PODS), 2002, pp. 271–281.
[9] H. Kaplan and T. Milo, *Short and simple labels for small distances and other functions*, in Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS), Lecture Notes in Comput. Sci., 2125, Springer, Berlin, 2001, pp. 246–257.
[10] H. Kaplan, T. Milo, and R. Shabo, *A comparison of labeling schemes for ancestor queries*, in Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 954–963.
[11] C. Li and T. W. Ling, *QED: A novel quaternary encoding to completely avoid re-labeling in XML updates*, in Proceedings of the Conference on Information and Knowledge Management (CIKM), 2005, pp. 501–508.
[12] C. Li, T. W. Ling, and M. Hu, *Efficient processing of updates in dynamic XML data*, in Proceedings of the International Conference on Data Engineering (ICDE), 2006.
[13] A. Marian, S. Abiteboul, G. Cobena, and L. Mignet, *Change-centric management of versions in an XML warehouse*, in Proceedings of the International Conference on Very Large Data Bases (VLDB), 2001, pp. 581–590.
[14] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, *Ordpaths: Insert-friendly XML node-labels*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 2004, pp. 903–908.
[15] N. Santoro and R. Khatib, *Labeling and implicit routing in networds*, Comput. J., 28 (1985), pp. 5–8.
[16] A. Silberstein, H. He, K. Yi, and J. Yang, *Boxes: Efficient maintenance of order-based labeling for dynamic XML data*, in Proceedings of the International Conference on Data Engineering (ICDE), 2005, pp. 285–296.
[17] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang, *Storing and querying ordered XML using a relational database system*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 2002, pp. 204–215.
[18] W3c, *Extensible Markup Language (XML)* 1.0, http://www.w3.org/TR/REC-xml.
[19] W3c, *Extensible Stylesheet Language (XSL)*, http://www.w3.org/Style/XSL/.
[20] W3c, *Xsl Transformations (XSLT) Specification*, http://www.w3.org/TR/WD-xslt.
[21] H. Wang, S. Park, W. Fan, and P. S. Yu, *ViST: A dynamic index method for querying XML data by tree structures*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 2003, pp. 110–121.
[22] X. Wu, M. L. Lee, and W. Hsu, *A prime number labeling scheme for dynamic ordered XML trees*, in Proceedings of the International Conference on Data Engineering (ICDE), 2004.
[23] XDEX, http://www.xmlindex.com.
[24] L. Xu, T. W. Ling, H. Wu, and Z. Bao, *DDE: From Dewey to a fully dynamic XML labeling scheme*, in Proceedings of the ACM SIGMOD International Conference on Management of Data, 2009, pp. 719–730.
[25] Xyleme, *A Dynamic Data Warehouse for the XML Data of the Web*, http://www.xyleme.com.
[26] A. C. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 222–227.
[27] U. Zwick and M. Thorup, *Compact routing schemes*, in Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), 2001, pp. 1–10.