SIAM J. DISCRETE MATH.
Vol. 19, No. 2, pp. 448–462

LABELING SCHEMES FOR SMALL DISTANCES IN TREES*

STEPHEN ALSTRUP†, PHILIP BILLE†, AND THEIS RAUHE†

Abstract. We consider labeling schemes for trees, supporting various relationships between nodes at small distance. For instance, we show that given a tree T and an integer k we can assign labels to each node of T such that given the label of two nodes we can decide, from these two labels alone, if the distance between v and w is at most k and, if so, compute it. For trees with n nodes and $k \geq 2$, we give a lower bound on the maximum label length of $\log n + \Omega(\log\log n)$ bits, and for constant k, we give an upper bound of $\log n + O(\log\log n)$. Bounds for ancestor, sibling, connectivity, and bi- and triconnectivity labeling schemes are also presented.

Key words. labeling schemes, trees

AMS subject classifications. 68R10, 68W01

DOI. 10.1137/S0895480103433409

1. Introduction.

Motivated by applications in XML search engines, network routing, and implicit graph representation, several *labeling schemes* for trees have been developed, among these [16, 22, 13, 10, 26, 1, 3, 8]. Given a tree, a labeling scheme assigns a *label*, which is a binary string, to each node v of the tree. Then, given only the labels of two nodes we can compute some predefined function of the two nodes. The main objective is to minimize the *maximum label length*, i.e., the maximum number of bits used in a label.

In this paper we consider labeling schemes for various relationships between nodes of small distance in trees. For instance, we show, by giving upper and lower bounds, that a labeling scheme supporting parent and sibling queries requires labels of length $\log n + \Theta(\log\log n)$.[1] This improves a recent bound by Kaplan and Milo [18] of $\log n + O(\sqrt{\log n})$.

More generally, we say that two nodes v and w with nearest common ancestor z are (k_1, k_2)-*related* if the distance from v to z is k_1 and the distance from w to z is k_2. For a positive integer k, a *k-relationship labeling scheme* is a labeling scheme for trees which supports tests for whether v and w are (k_1, k_2)-related for all nodes v and w and all positive integers $k_1, k_2 \leq k$. In particular, a 1-relationship labeling scheme supports tests for whether two nodes are $(0,0)$-, $(0,1)$-, $(1,0)$-, or $(1,1)$-related, that is, whether two nodes are identical, one is the parent of the other, or they are siblings. For trees with n nodes we show, for $k = 1$, a lower bound on the label length of $\log n + \Omega(\log\log n)$, and for fixed, constant k we give an upper bound of $\log n + O(\log\log n)$.

As noted in [18], a k-relationship labeling scheme can be used to test whether the distance between two nodes is at most k, and if this is the case we can compute the distance exactly. We call a labeling scheme with this property a *k-restricted distance labeling scheme*. We give a lower bound showing that for $k = 2$, a k-restricted

*Received by the editors August 15, 2003; accepted for publication (in revised form) November 29, 2004; published electronically November 4, 2005. An extended abstract of this paper appeared in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2003.
http://www.siam.org/journals/sidma/19-2/43340.html

†IT University of Copenhagen, Rued Langgardsvej 7, DK-2300 Copenhagen S, Denmark (stephen@itu.dk, beetle@itu.dk, theis@itu.dk).

[1] log refers to the binary logarithm and $\log^*$ is the number of times log should be iterated to get a constant.

distance labeling scheme requires labels of length $\log n + \Omega(\log \log n)$. Hence, for constant k, our k-relationship labeling scheme gives a k-restricted distance labeling scheme which is optimal to within a factor of $\log \log n$. This result improves a recent upper bound of $\log n + O(\sqrt{\log n})$ for k-relationship and k-restricted distance labeling schemes given in [18]. In contrast to the results for restricted distances, Gavoille et al. [13] show that a labeling scheme for computing the distance between any pair of nodes in a tree must use labels of length $\Theta(\log^2 n)$. In [10] it is shown that even if the distances are allowed to be approximated to within a factor of $(1 + 1/\log n)$ we still need labels of length $\Theta(\log n \log \log n)$. Our result shows that for restricted distances much smaller labels suffice. A 1-restricted labeling scheme supports tests for whether two nodes are identical or adjacent. Such a labeling scheme, called an *adjacency labeling scheme*, was recently given for trees in [4], with label length bounded by $\log n + O(\log^* n)$. Thus, there is a provable gap between the label length of 1- and 2-restricted distance labeling schemes.

The above lower bounds are the result of a more general new technique which we use to obtain lower bounds for several types of labeling schemes, and for many of these we give matching upper bounds. Apart from the above results we present the following.

1.1. Bi- and triconnectivity labeling schemes. As an application of our k-relationship labeling scheme we obtain a labeling scheme for general graphs for biconnectivity (or 2-vertex connectivity) queries. Recently, Katz et al. [21] considered labeling schemes for 1-, 2-, 3-, and m-vertex connectivity. They gave a labeling scheme for biconnectivity using $3 \log n$ bits. We show, giving upper and lower bounds, that labels of length $\log n + \Theta(\log \log n)$ are required. The labeling scheme for triconnectivity (or 3-vertex connectivity) in [21] uses the biconnectivity labeling scheme and has label length bounded by $5 \log n$. Using our biconnectivity labeling scheme we obtain a triconnectivity labeling scheme using labels of length $3 \log n + O(\log \log n)$.

1.2. Ancestor labeling schemes. For trees with n nodes we show that a labeling scheme for ancestor queries must use labels of length $\log n + \Omega(\log \log n)$. This is the first nontrivial lower bound for the problem. Upper bounds using $2 \lceil \log n \rceil$ bits were given in [27, 17, 23]. Recently, Abiteboul, Kaplan, and Milo [1] gave an ancestor labeling scheme using labels of length $3/2 \log n + O(\log \log n)$. Subsequently, this was improved by Alstrup and Rauhe [3], bounding the label length to $\log n + O(\sqrt{\log n})$.

If no two nodes are assigned to the same label, we say that the labels are unique. The above labeling schemes all produce unique labels, whereas the lower bounds also hold for labeling schemes that produce nonunique labels. However, the following bounds show that there is a nontrivial complexity difference between labeling schemes assigning unique and nonunique labels.

1.3. Sibling and connectivity labeling schemes. For sibling queries we give a labeling scheme using labels of length $\lceil \log n \rceil$. This labeling scheme will not assign unique labels to the nodes of the tree. For the case where uniqueness is required, as in [16], we give upper and lower bounds of $\log n + \Theta(\log \log \Delta)$ for trees of maximum degree Δ. Extending the result for the sibling labeling scheme we give a labeling scheme supporting connectivity queries for forests of n nodes using labels of length $\lceil \log n \rceil$. Again, these labels are not unique, and if uniqueness is required we show that labels of length $\log n + \Theta(\log \log n)$ are required.

1.4. Related work. Adjacency labeling schemes were introduced by Breuer and Folkman [5, 6], and efficient labeling schemes were considered by Kannan, Naor, and

Rudich in [16, 17]. In [22] *distance labeling schemes* were introduced, i.e., labeling schemes that compute the distance between any pair of nodes. Distance labeling schemes for various types of graphs are given in [22, 20, 13, 11], and distance labeling schemes computing approximate distances are given in [10, 25].

Recently, labeling schemes for various other relationships have been studied. Labeling schemes are given for ancestor in [17, 1, 26, 3, 19, 8], for nearest common ancestor in [2], and for connectivity in [21]. Efficient labeling schemes are also applicable to routing schemes; see, e.g., [23, 26]. A survey on labeling schemes can be found in [12].

1.5. Outline. In section 2 we give some preliminaries, and in sections 3, 4, and 5 we present the upper bounds on relationship, bi- and triconnectivity, connectivity, and sibling labeling schemes. Lower bounds for these schemes are shown in section 6 together with lower bounds for ancestor labeling schemes and the above-mentioned lower bound technique.

2. Preliminaries. For a graph G we denote the set of nodes and edges by $V(G)$ and $E(G)$. Let T be a rooted tree with n nodes. The degree of a node $v \in V(T)$, $\deg(v)$, is the number of children of v and the degree of T, $\deg(T)$, is given by $\deg(T) = \max_{v \in V(T)} \deg(v)$. Note that an edge $(v, \text{parent}(v))$ does not contribute to $\deg(v)$. The distance between two nodes $v, w \in V(T)$, denoted by $\text{dist}(v, w)$, is the number of edges on the unique simple path between v and w. The depth of v is the distance between v and the root of T. We let $T(v)$ denote the subtree of T rooted at a node $v \in V(T)$. If $w \in V(T(v))$, then v is an ancestor of w, and if $w \in V(T(v)) \backslash \{v\}$, then v is a proper ancestor of w. If v is (proper) ancestor of w, then w is a (proper) descendant of v. A node z is a common ancestor of v and w if it is an ancestor of v and w. The nearest common ancestor of v and w, $\text{nca}(v, w)$, is the common ancestor of v and w of largest depth. For a node v of depth d and $i \leq d$, the ith *level ancestor* of v, $A(v, i)$, is the ancestor of v of depth $d - i$. We call the nodes $A(v, 1)$ and $A(v, 2)$ the parent (denoted $\text{parent}(v)$) and grandparent of v, respectively. Two nodes are siblings if they have the same parent. A node with no children is a leaf and otherwise is an internal node. Two nodes in a forest are connected if and only if there is a path between them. A bit string of length n is a sequence $a = a_0 a_1 \dots a_{n-1}$, where $a_i \in \{0, 1\}$, $0 \leq i \leq n-1$. For $0 \leq j \leq n-1$ the sequences $a_0, \dots, a_{j-1}$ and $a_{n-j}, \dots, a_{n-1}$ are the j *most significant bits* and the j *least significant bits*, respectively. The *standard binary representation* of a positive integer k is the unique bit string $a_0 \dots a_{r-1}$, where $r = \lceil \log k \rceil$ and $k = \sum_{j=0}^{r-1} a_j 2^{r-j-1}$. The *discrete logarithm* of k is the number $\lfloor \log k \rfloor$. For two integers i and j, where $i \leq j$, let $[i, j]$ be the interval $\{i, \dots, j\}$.

2.1. Labeling schemes. A *binary query* (or simply query) is a mapping $f : V(G) \times V(G) \to X$ for some set X. A *labeling scheme* for a family of graphs $\mathcal{F}$ supporting queries $f_1, \dots, f_m$ ($f_i : V(G) \times V(G) \to X_i$) is a tuple $(e, d_1, \dots, d_m)$ of mappings, where e is called the *encoder* and d_i is called the decoder for the ith query. The encoder e defines a *label assignment*, e_G, for all $G \in \mathcal{F}$, which is a mapping of $V(G)$ into bit strings called *labels*. Given the labels of two nodes v and w, the ith decoder, d_i, computes the ith query, i.e., $d_i(e_G(v), e_G(w)) = f_i(v, w)$. If the label assignment e_G is an injective mapping for all $G \in \mathcal{F}$, we say that the labeling scheme assigns *unique* labels to the nodes. A labeling scheme has *label length* bounded by s if the maximum length of the labels assigned to a node in any $G \in \mathcal{F}$ is bounded by s. We say that a labeling scheme can be computed in time t if there is an encoder e

such that for any $G \in \mathcal{F}$, e assigns labels to all nodes in G in time t.

3. Upper bound for relationship labeling schemes.

3.1. A 1-relationship labeling scheme. In this section we give a 1-relationship labeling scheme, which will serve as a basis for our k-relationship labeling scheme in the next section. As a consequence, some of the lemmas shown below will be more general than required for a 1-relationship labeling scheme. Our labeling scheme assigns unique labels to each node and supports both parent and sibling queries. As described, a labeling scheme with these properties implies a 1-relationship labeling scheme. The labeling scheme has label length bounded by $\log n + O(\log \log n)$ for trees with n nodes.

Some of the ideas in this section are inspired by [4]. There a simple labeling scheme supporting parent (but not sibling) queries is given with labels of length bounded by $\log n + O(\log \log n)$. Subsequently, they use this result to construct a more complicated labeling scheme with labels of length bounded by $\log n + O(\log^* n)$. In this section we instead generalize the simple labeling scheme supporting parent queries to also handle sibling queries within the same bounds. As noted in the introduction we later show that our labels are the smallest possible within a factor of $\log \log n$.

Let $\mathcal{T}_n$ denote the family of rooted trees with n nodes. Let $T \in \mathcal{T}_n$. As in [14] we partition T into disjoint paths. For a node $v \in V(T)$ let $\text{size}(v) = |V(T(v))|$. We classify each node of T as either *heavy* or *light* as follows. The root is light. For each internal node v we pick a child w of v of maximum size among the children of v and classify w as heavy. The remaining children are light. We call an edge to a light child a *light edge* and an edge to a heavy child a *heavy edge*. For an internal node v, let $\text{heavy}(v)$ denote the heavy child of v. Define the *light subtree*, $L(w)$, rooted at the node w as follows. If w is an internal node, then $L(w)$ is the subtree obtained from $T(w)$ by cutting away $T(\text{heavy}(w))$, and if w is a leaf $L(w) = T(w)$. Let $\text{lightsize}(v) = |V(L(v))|$. The *light depth* of a node v, $\text{lightdepth}(v)$, is the number of light edges on the path from v to the root.

LEMMA 1 (see [14]). *For any tree T with n nodes* $\text{lightdepth}(v) \leq \log n + O(1)$ *for any $v \in T$.*

The nearest light ancestor of v (possibly v itself) is denoted $\text{apex}(v)$. By removing the light edges T is partitioned into *heavy paths*.

A key ingredient of the scheme is *preorder numbers*. Order the tree T such that the rightmost child of each internal node is the heavy node. The light children need not be in any particular order. A *preorder depth first traversal* of T is obtained by first visiting the root and then recursively visiting the children of the root from left to right. The *preorder number*, $\text{pre}(v)$, is the number of nodes visited before v in this traversal, i.e., the root will have number 0 and the rightmost leaf will have number $n-1$. The labels assigned by our labeling scheme will encode $\text{pre}(v)$ in the label of v using $\lceil \log n \rceil$ bits. This will ensure that the labels are unique. In the rest of the label we will encode various smaller fields using no more than $O(\log \log n)$ bits in total. In the following we show how to test, for two nodes v and w, if one is the parent of the other or if they are siblings based on whether v and w are light or heavy nodes.

First define a node w to be a *significant ancestor* of v if $v \in L(w)$. Note that a node is its own significant ancestor. We have the following relation between significant ancestors and the preorder numbering.

LEMMA 2. *For all nodes v and w, $v \in L(w)$ if and only if* $\text{pre}(v) \in [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$.

Proof. If w is a leaf, then $v = w$ and $\text{lightsize}(w) = 1$. Hence, $\text{pre}(w) = \text{pre}(v) = \text{pre}(w)+\text{lightsize}(w)-1$ and the result follows. So assume w is an internal node. Then, in a preorder traversal, v is visited at the time of w or after and before $\text{heavy}(w)$ if and only if $\text{pre}(w) \leq \text{pre}(v) < \text{pre}(\text{heavy}(w))$. Since $\text{pre}(\text{heavy}(w)) = \text{pre}(w)+\text{lightsize}(w)$ the result follows. ∎

Consider the binary representation of $\text{pre}(v)$ for an internal node v. Let $f(v) = \lfloor \log \text{lightsize}(v) \rfloor$. We define the *significant preorder number*, $\text{spre}(v)$, as the smallest number greater than or equal $\text{pre}(v)$ which is a multiple of $2^{f(v)}$. Equivalently,

$$\text{spre}(v) = \begin{cases} \text{pre}(v) & \text{if } \text{pre}(v) \bmod 2^{f(v)} = 0, \\ \text{pre}(v) - (\text{pre}(v) \bmod 2^{f(v)}) + 2^{f(v)} & \text{otherwise.} \end{cases}$$

The following lemma states the relations we need between the preorder and significant preorder numbers.

LEMMA 3. *For all nodes v and w the following hold:*

(i) $\text{spre}(v) \in [\text{pre}(v), \text{pre}(v) + \text{lightsize}(v) - 1]$.

(ii) $v = w$ *if and only if* $\text{lightdepth}(v) = \text{lightdepth}(w)$ *and* $\text{spre}(v) = \text{spre}(w)$.

(iii) *If* $\text{lightdepth}(v) = \text{lightdepth}(w)$, *then* $\text{pre}(w) < \text{pre}(v)$ *if and only if* $\text{spre}(w) < \text{spre}(v)$.

Proof. (i) If $\text{pre}(v) \bmod 2^{f(v)} = 0$, then $\text{spre}(v) = \text{pre}(v)$, and since $\text{lightsize}(v) \geq 1$ for all v the result follows. Otherwise $1 \leq \text{pre}(v) \bmod 2^{f(v)} \leq 2^{f(v)} - 1$. Hence, $\text{spre}(v) \geq \text{pre}(v) - (2^{f(v)} - 1) + 2^{f(v)} = \text{pre}(v) + 1$ and $\text{spre}(v) \leq \text{pre}(v) - 1 + 2^{f(v)} \leq \text{pre}(v) - 1 + \text{lightsize}(v)$.

(ii) If $v = w$, the conditions are clearly satisfied. Conversely, assume that $v \neq w$ and the conditions are satisfied. Since $v \neq w$ and $\text{lightdepth}(v) = \text{lightdepth}(w)$ we have that $v \notin L(w)$ and $w \notin L(v)$. Then, by Lemma 2 $\text{pre}(v) \notin [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ and $\text{pre}(w) \notin [\text{pre}(v), \text{pre}(v) + \text{lightsize}(v) - 1]$, and hence these intervals must be disjoint. However, since $\text{spre}(v) = \text{spre}(w)$ we have, by (i), the contradiction that $\text{spre}(v) \in [\text{pre}(v), \text{pre}(v) + \text{lightsize}(v) - 1]$ and $\text{spre}(v) \in [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$.

(iii) Assume that $\text{lightdepth}(v) = \text{lightdepth}(w)$. If $\text{pre}(w) < \text{pre}(v)$, then $v \notin L(w)$. By Lemma 2, $\text{pre}(v) \notin [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ and since $\text{pre}(w) < \text{pre}(v)$ we have $\text{pre}(w) + \text{lightsize}(w) - 1 < \text{pre}(v)$. By (i) it follows that $\text{spre}(w) < \text{spre}(v)$. Conversely, since $\text{spre}(w) < \text{spre}(v)$ and $\text{lightdepth}(v) = \text{lightdepth}(w)$ we have by (ii) that $v \neq w$. Furthermore, as in the proof of (ii), this implies that the intervals $[\text{pre}(v), \text{pre}(v) + \text{lightsize}(v) - 1]$ and $[\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ are disjoint. By (i), $\text{spre}(v) \in [\text{pre}(v), \text{pre}(v) + \text{lightsize}(v) - 1]$ and $\text{spre}(w) \in [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ and since these intervals are disjoint and $\text{spre}(w) < \text{spre}(v)$ we have that $\text{pre}(w) < \text{pre}(v)$. ∎

Note that by Lemma 3(ii) it follows that any node v is uniquely identified by $\text{spre}(v)$ and $\text{lightdepth}(v)$. The following lemma shows that the significant preorder number of a significant ancestor can be represented efficiently. In particular, $\text{spre}(\text{parent}(v))$ can be represented efficiently if v is a light node.

LEMMA 4. *Given* $\text{pre}(v)$ *we can represent* $\text{spre}(w)$ *for each significant ancestor w of v using only* $\log \log n + O(1)$ *bits per significant ancestor.*

Proof. Let w be a significant ancestor of v. Since $\text{lightsize}(w) < 2^{f(w)+1}$ there can be, apart from $\text{spre}(w)$, at most one other number in the interval $[\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ with all the $f(w)$ least significant bits set to zero, i.e., the number $\text{spre}(w) + 2^{f(w)}$. Let $\text{pre}'(v)$ be $\text{pre}(v)$ with all the $f(w)$ least significant bits set to zero. Since w is a significant ancestor of v, $v \in L(w)$ and thus, by Lemma 2, $\text{pre}(v) \in$

$[\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$. Hence, $\text{pre}'(v)$ will be either $\text{spre}(w) - 2^{f(w)}$, $\text{spre}(w)$ or $\text{spre}(w) + 2^{f(w)}$ and therefore $\text{spre}(w)$ is either $\text{pre}'(v) + 2^{f(w)}$, $\text{pre}'(v)$, or $\text{pre}'(v) - 2^{f(w)}$. Clearly, representing $f(w)$ and two extra bits to distinguish these three cases we can compute $\text{spre}(w)$ from $\text{pre}(v)$. This can be represented by $\lceil \log \log n \rceil + 2$ bits since $f(w)$ is bounded by $\log n$. □

For each light node v we will encode $\text{lightdepth}(v)$, $\text{spre}(v)$, and $\text{spre}(\text{parent}(v))$ in the label of v. By Lemma 1 $\text{lightdepth}(v) \leq \log n + O(1)$ and can thus be represented using $\log \log n + O(1)$ bits. Since the labels encode $\text{pre}(v)$ and v is light, we have by Lemma 4 that $\text{spre}(v)$ and $\text{spre}(\text{parent}(v))$ can also be represented using $\log \log n + O(1)$ bits. By Lemma 3(ii), $\text{lightdepth}(v)$ together with $\text{spre}(v)$ uniquely identifies the node v. This immediately implies the following.

LEMMA 5. *For a light node v and internal node w, w is the parent of v if and only if* $\text{lightdepth}(v) = \text{lightdepth}(w) + 1$ *and* $\text{spre}(\text{parent}(v)) = \text{spre}(w)$.

LEMMA 6. *For two light nodes v and w, w and v are siblings if and only if* $\text{lightdepth}(v) = \text{lightdepth}(w)$ *and* $\text{spre}(\text{parent}(v)) = \text{spre}(\text{parent}(w))$.

Next we show how to handle the remaining cases. Define $\text{diff\_parent}(v) = \text{spre}(v) - \text{spre}(\text{parent}(v))$ and leave it undefined for the root. Similarly, for internal nodes, define $\text{diff\_heavy}(v) = \text{spre}(\text{heavy}(v)) - \text{spre}(v)$ and leave it undefined for leaves. The following lemma shows how the discrete logarithm of $\text{diff\_parent}(v)$ and $\text{diff\_heavy}(v)$ can be used to test for parenthood between two nodes on a heavy path. Since the discrete logarithm is bounded by $\log n$, only $\lceil \log \log n \rceil$ bits are needed to represent each of these numbers.

LEMMA 7. *For heavy node v and internal node w, w is the parent of v if and only if* $\text{spre}(w) < \text{spre}(v)$, $\text{lightdepth}(v) = \text{lightdepth}(w)$ *and* $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_parent}(v) \rfloor = \lfloor \log \text{diff\_heavy}(w) \rfloor$

Proof. For $w = \text{parent}(v)$ it is straightforward, using Lemma 3, to verify that the conditions are satisfied. Conversely, assume that a node $w \neq \text{parent}(v)$ satisfies the conditions. Since $\text{spre}(w) < \text{spre}(v)$ and $\text{lightdepth}(v) = \text{lightdepth}(w)$, we have by Lemma 3(iii) that $\text{pre}(w) < \text{pre}(v)$, and therefore w cannot be a descendant of v. Furthermore, w cannot be a descendant of any other sibling of v, because then $\text{lightdepth}(w) > \text{lightdepth}(v)$. It follows that w cannot be a descendant of $\text{parent}(v)$. Hence, in a preorder traversal of T the node $\text{heavy}(w)$ is visited before $\text{parent}(v)$ or $\text{heavy}(w) = \text{parent}(v)$. That is, $\text{pre}(\text{heavy}(w)) \leq \text{pre}(\text{parent}(v))$ and by Lemma 3(ii) and (iii), also $\text{spre}(\text{heavy}(w)) \leq \text{spre}(\text{parent}(v))$, and therefore $\text{spre}(v) - \text{spre}(w) \geq (\text{spre}(\text{heavy}(w)) - \text{spre}(w)) + (\text{spre}(v) - \text{spre}(\text{parent}(v))) = \text{diff\_heavy}(w) + \text{diff\_parent}(v)$. By the identities $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_parent}(v) \rfloor = \lfloor \log \text{diff\_heavy}(w) \rfloor$ this leads to the contradiction $\text{spre}(v) - \text{spre}(w) \geq \text{diff\_heavy}(w) + \text{diff\_parent}(v) \geq 2 \cdot 2^{\lfloor \log \text{diff\_parent}(v) \rfloor} = 2 \cdot 2^{\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor} > \text{spre}(v) - \text{spre}(w)$. □

Considering siblings instead, we immediately obtain the following corollary to Lemma 7.

LEMMA 8. *A heavy node v and light node w are siblings if and only if* $\text{spre}(\text{parent}(w)) < \text{spre}(v)$, $\text{lightdepth}(v) = \text{lightdepth}(w) - 1$, *and* $\lfloor \log(\text{spre}(v) - \text{spre}(\text{parent}(w))) \rfloor = \lfloor \log \text{diff\_parent}(v) \rfloor = \lfloor \log \text{diff\_heavy}(\text{parent}(w)) \rfloor$.

Note that since any node has at most one heavy child, two heavy nodes v and w are siblings if and only if $v = w$. Since the labels are unique it is trivial to handle this case.

Combining the above lemmas we obtain the 1-relationship labeling scheme. For $T \in \mathcal{T}_n$ let the encoder $e_T(v), v \in V(T)$, encode $\text{pre}(v)$, $\text{lightdepth}(v)$, $\text{spre}(v)$,

$\lfloor \log \text{diff\_heavy}(v) \rfloor$ and a type bit indicating if v is a light or heavy node. Furthermore, if v is a light node encode $\text{spre}(\text{parent}(v))$ and $\lfloor \log \text{diff\_heavy}(\text{parent}(v)) \rfloor$. If v is a heavy node encode $\lfloor \log \text{diff\_parent}(v) \rfloor$. As described, $\text{pre}(v)$ uses $\lceil \log n \rceil$ bits and each of the other values uses $\log \log n + O(1)$ bits each. For easy decoding we represent each of the values in fixed sized fields in the label of v. The first $\lceil \log n \rceil$ bits stores $\text{pre}(v)$. The other values are represented, in five fields (we leave one field undefined when v is a light node) of the same length, in the next $5 \log \log n + O(1)$ bits. At the end of the label we store the type bit. We will assume that the decoder does not know the value n, i.e., the decoder is not specialized to trees of size n but will work with any tree, regardless of its size. Due to this restriction we cannot compute $\lceil \log n \rceil$ directly and use this to extract the preorder number and then the rest of the fields. Instead we use a self-delimiting code for $\lceil \log n \rceil$. In particular, we prefix the label with $1^{|x|}0x$, where x is the binary representation of the length of the field containing $\text{pre}(v)$. Since the length of $\text{pre}(v)$ is $\lceil \log n \rceil$, we have added only $2 \log \log n + O(1)$ bits. Note that the unary prefix $1^{|x|}0$ enables us to figure out the length of x. In total the label length will be bounded by $\log n + O(\log \log n)$. By uniqueness of the labels and Lemmas 5 through 8, it is straightforward to construct decoders testing if two nodes are $(0,0)$-, $(0,1)$-, $(1,0)$-, or $(1,1)$-related. In summary we have the next theorem.

THEOREM 1. *For trees with n nodes there is a 1-relationship labeling scheme with label length bounded by $\log n + O(\log \log n)$.*

Finally, note that labels for all nodes in T can be computed in $O(n)$ time and queries can be implemented in $O(1)$ time per query assuming standard binary operations on a RAM.

3.2. A general k-relationship labeling scheme. In this section we generalize the result of the previous section to a k-relationship labeling scheme. The scheme extends the ideas of the first labeling scheme and has label length bounded by $\log n + O(k^2(\log \log n + \log k))$, which for constant k is $\log n + O(\log \log n)$.

We first extend the definition of $\text{diff\_heavy}(v)$ and $\text{diff\_parent}(v)$ as follows. If v has a descendant u on the same heavy path as v of distance m, let $\text{diff\_heavy}(v,m) = \text{spre}(u) - \text{spre}(v)$, and if there is no such node u let $\text{diff\_heavy}(v,m) = 2n$, i.e., the discrete logarithm of $2n$ will be $\lfloor \log n \rfloor + 1$, indicating that this is not an actual difference. Similarly, define $\text{diff\_parent}(v,m)$ for the ancestor on the same heavy path of v of distance m. Furthermore, for a node v we define the *index* of v, $\text{index}(v)$, as the number of nodes with the same light depth as v and with smaller preorder numbers than v. We will use the following generalization of Lemma 7.

LEMMA 9. *For a heavy node v and internal node w, w and v are on the same heavy path and w is an ancestor of v of distance $m \geq 1$ if and only if $\text{spre}(w) < \text{spre}(v)$, $\text{lightdepth}(v) = \text{lightdepth}(w)$, $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_parent}(v,m) \rfloor = \lfloor \log \text{diff\_heavy}(w,m) \rfloor$, and $\text{index}(v) \bmod m = \text{index}(w) \bmod m$.*

Proof. Let x denote the ancestor of v of distance m on the heavy path of v. Similarly, let y denote the descendant of w of distance m on the heavy path of w. If x or y does not exist, then the conditions do not hold by definition of diff_parent and diff_heavy. If they both exist and if $x = w$ (or equivalently $y = v$), it is straightforward to check that the conditions are satisfied. Conversely, assume that the conditions are satisfied and $x \neq w$. Since $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_parent}(v,m) \rfloor = \lfloor \log \text{diff\_heavy}(w,m) \rfloor$, both x and y exist and are on the same heavy paths as v and w, respectively. As in the proof of Lemma 7, we have $\text{lightdepth}(w) = \text{lightdepth}(y) = \text{lightdepth}(x) = \text{lightdepth}(v)$ and $\text{pre}(w) < \text{pre}(v)$.

Since $\text{index}(w) \bmod m = \text{index}(y) \bmod m = \text{index}(x) \bmod m = \text{index}(v) \bmod m$ and $x \neq w$, the paths from w to y and x to v are either disjoint or $x = y$. Thus $\text{pre}(y) \leq \text{pre}(x)$ and by Lemma 3(ii) and (iii) also $\text{spre}(y) \leq \text{spre}(x)$. Therefore $\text{spre}(v) - \text{spre}(w) \geq \text{diff\_heavy}(w, m) + \text{diff\_parent}(v, m)$. By the identities $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_parent}(v, m) \rfloor = \lfloor \log \text{diff\_heavy}(w, m) \rfloor$, we obtain the contradiction $\text{spre}(v) - \text{spre}(w) \geq \text{diff\_heavy}(w, m) + \text{diff\_parent}(v, m) \geq 2 \cdot 2^{\lfloor \log \text{diff\_parent}(v,m) \rfloor} = 2 \cdot 2^{\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor} > \text{spre}(v) - \text{spre}(w)$. □

The main idea in our labeling scheme is to store, in the label of v, $\text{pre}(v)$ and $\text{lightdepth}(v)$ as before. Furthermore, for each significant ancestor w of v of distance at most k we will represent $\text{spre}(w)$ together with $\text{diff\_heavy}(w, m)$, $\text{diff\_parent}(w, m)$, and $\text{index}(w) \bmod m$ for $1 \leq m \leq k$. Then, to test if two nodes v and w are (k_1, k_2)-related we identify the heavy path containing the nearest common ancestor of v and w and compute distances to and on this heavy path.

3.3. The encoder. We can now describe the encoder for our k-relationship labeling scheme. For $T \in \mathcal{T}_n$ let the label $e_T(v)$, $v \in V(T)$ encode $\text{pre}(v)$ and $\text{lightdepth}(v)$. Furthermore, we store an *ancestor table* of s entries, where s is the number of significant ancestors of distance at most k from v. If w is the ith significant ancestor of v, the ith entry in the ancestor table will represent $\text{spre}(w)$, $\text{dist}(v, w)$, and a single bit, called the *apex bit*, indicating whether the distance $\text{dist}(w, \text{apex}(w))$ is at most k. If this is so we store $\text{dist}(w, \text{apex}(w))$ and otherwise leave this field undefined. Furthermore, the ith entry also represents, for $1 \leq m \leq k$, $\text{diff\_heavy}(w, m)$, $\text{diff\_parent}(w, m)$ and $\text{index}(w) \bmod m$. Hence, number of bits used to represent an entry is bounded by $O(k \log \log n + k \log k)$ and thus the total number of bits used for the ancestor table is at most $O(k^2(\log \log n + \log k))$. Note that since w is the ith significant ancestor we have that $\text{lightdepth}(w) = \text{lightdepth}(v) - i$ and hence this information is implictly stored in the table.

For efficient computation of the queries we store a *lookup table* of k entries. The ith entry stores the light depth of $A(v, i)$. Hence the lookup table uses at most $O(k \log \log n)$ bits. As before all the values are stored in fixed sized fields and we prefix the label with small codes representing the length of $\text{pre}(v)$ and each of tables. In total the label length is bounded by $\log n + O(k^2(\log \log n + \log k))$. Computing the tables can be done in $O(k)$ time per node after $O(n)$ time preprocessing and hence the labeling scheme can be computed in $O(nk)$ time.

3.4. The decoder. In the following we present the decoder for our k-relationship labeling scheme. We first present necessary and sufficient conditions for two nodes v and w to be (k_1, k_2)-related and then show how to test these conditions using only the labels of v and w.

LEMMA 10. *Let $v, w \in T$ and distances k_1 and k_2 (not both zero) be given. Let v' be the significant ancestor of v such that* $\text{lightdepth}(v') = \text{lightdepth}(A(v, k_1))$ *and if $v' \neq v$ let v'' be the significant ancestor of v of light depth* $\text{lightdepth}(A(v, k_1)) + 1$. *Otherwise let $v'' = v$. Similarly, define w' and w'' for w and k_2. Then, v and w are (k_1, k_2)-related if and only if one of the following disjoint conditions is satisfied:*

(i) *$v' = w'$, v'', and w'' are on different heavy paths,* $\text{dist}(v, v') = k_1$ *and* $\text{dist}(w, w') = k_2$.

(ii) *v' and w' are on same heavy path, v' is a proper ancestor of w',* $\text{dist}(w', v') = k_2 - \text{dist}(w, w')$, *and* $\text{dist}(v, v') = k_1$.

(iii) *v' and w' are on same heavy path, w' is a proper ancestor of v',* $\text{dist}(v', w') = k_1 - \text{dist}(v, v')$, *and* $\text{dist}(w, w') = k_2$.

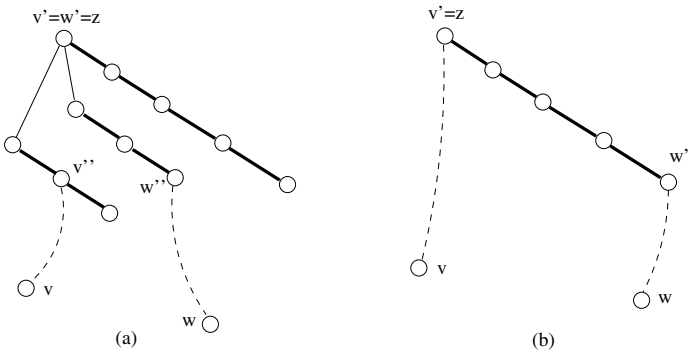

FIG. 1. *Cases for Lemma* 10: (a) *case* (i), (b) *case* (ii).

Proof. The situation is illustrated in Figure 1. Let $z = \text{nca}(v, w)$. If one of the conditions is satisfied it is straightforward to check that v and w are (k_1, k_2)-related. Conversely, if v and w are (k_1, k_2)-related, then z must be on the heavy path of v' and w' and $z = v'$ or $z = w'$. If $z = v' = w'$, then z is a significant ancestor of both v and w. Hence, since k_1 and k_2 are not both zero, v'' and w'' must be on different heavy paths; otherwise there would be a common ancestor of larger depth than z contradicting the assumption that $z = \text{nca}(v, w)$. If $z = v' \neq w'$, then v' is a proper ancestor of w', and if $z = w' \neq v'$, then w' is a proper ancestor of v'. Since v and w are (k_1, k_2)-related the distance conditions are satisfied. □

Given only the labels of the nodes v and w we can test if they are (k_1, k_2)-related for $k_1, k_2 \leq k$ as follows. First, since the labels are unique, it is trivial to test if v and w are $(0, 0)$-related. Hence, we will assume that k_1 and k_2 are not both zero. We will show how to test each of the conditions in Lemma 10 using only the labels. Using the lookup tables we first compute the entries in the ancestor tables for the nodes v', v'', w', and w''. Assume that the values stored at these entries of the tables are available. Using Lemma 3(ii) we can check if $v' = w'$. The distances $\text{dist}(v, v')$ and $\text{dist}(w, w')$ are stored directly in the ancestor tables of v and w, and the first three conditions in (ii) and (iii) can be checked using Lemma 9. What remains is to describe how to test if v'' and w'' are on different heavy paths. Since the distances $\text{dist}(v'', \text{apex}(v''))$ and $\text{dist}(w'', \text{apex}(w''))$ are both smaller than k, they are available in the ancestor tables. If v'' and w'' are on the same heavy path their distance must be $|\text{dist}(v'', \text{apex}(v'')) - \text{dist}(w'', \text{apex}(w''))|$, so we can use Lemma 9 to test whether they are on the same heavy path and if so if they are this distance apart. Thus we have shown that the conditions of Lemma 10 can be tested using only the labels of v and w and so we can determine if v and w are (k_1, k_2)-related. In summary we have shown the next theorem.

THEOREM 2. *For trees with n nodes there is a k-relationship labeling scheme with label length bounded by* $\log n + O(k^2(\log\log n + \log k))$.

As noted, the k-relationship labeling scheme can be computed in $O(nk)$ time and due to the lookup and ancestor tables queries can be performed in $O(1)$ time.

4. Upper bounds for bi- and triconnectivity labeling schemes. As an application of our k-relationship labeling scheme of section 3 we give a labeling scheme for biconnectivity. Subsequently, we use a reduction from [21] to obtain a labeling scheme for triconnectivity. Both labeling schemes assign unique labels. For a graph G with n nodes, the labeling scheme for bi- and triconnectivity uses labels of length

bounded by $\log n + O(\log \log n)$ and $3 \log n + O(\log \log n)$, respectively.

We first give some preliminaries. Let G be a graph. A set of paths P connecting two nodes v and w in G is *vertex-disjoint* if each node except v and w appears in at most one path $p \in P$. We define v and w to be *m-vertex connected* if there is a set of vertex-disjoint paths of size m connecting v and w. We say that v and w are bi- or triconnected if they are 2- or 3-vertex connected, respectively. A *cut-node* is a node whose removal (and all incident edges) disconnects the graph. A *block* of a graph G is a maximal connected subgraph without a cut-node. By maximality, different blocks of G overlap in at most one node, which is then the cut-node. Using Menger's theorem (see, e.g., [9]), it can be shown that two nodes $v, w \in V(G)$ are biconnected if and only if they are within the same block and the block has at least three nodes.

We define the *block graph* B of G. Each node in G is represented by a unique node in B and each node in B either represents a node in G or a block with at least three nodes in G. The edges of B are defined as follows. Let v be a node in G and let $B(v)$ denote the set of blocks in G that contain v and have at least three nodes. For each node representing a block $b \in B(v)$ there is an edge to the node representing v in B. A node in B representing a node in G that is not contained in any block with at least three nodes is not incident to any other node in B. By the maximality of blocks we have the next lemma.

LEMMA 11. *The block graph B of a graph G is a forest of unrooted trees.*

Using depth-first search [24], we can compute the block forest in linear time. We root each tree in the forest as follows. If the tree contains only one node, this node is the root. Otherwise the tree contains at least one node representing a block and we arbitrarily root the tree in such a node. By B_r we denote the rooted version of the block forest B.

LEMMA 12. *Two nodes v and w in G are biconnected in G if and only if, in the block forest of rooted trees B_r, either v and w are siblings, v is the grandparent of w, or w is the grandparent of v.*

Proof. If v and w are biconnected in G, then they are contained in the same block with at least three nodes, and hence they are incident to the same node representing a block. In B_r, this implies that v and w are either siblings or one is the grandparent of the other. Conversely, if v and w are siblings or one is the grandparent of the other in B_r, then they are incident to the same node representing a block. Hence, they are contained in the same block with at least three nodes and are thus biconnected. ∎

To test the conditions in Lemma 12 we extend our k-relationship labeling scheme to handle the more general case of forests. Add an extra root node connected to each root of the trees in the forest. This produces a tree where we then apply our k-relationship labeling scheme. The modifications needed to handle a special root node are straightforward to implement. Using a 2-relationship labeling scheme for the forest B_r we obtain by Lemma 12 the following theorem.

THEOREM 3. *For graphs with n nodes there is a biconnectivity labeling scheme that assigns unique labels with label length bounded by $\log n + O(\log \log n)$.*

Since we can compute the block forest B_r in $O(n)$ time, the labeling scheme can be computed in $O(n)$ time and with the 2-relationship labeling scheme queries can be answered in $O(1)$ time.

As noted in the introduction we can use our biconnectivity labeling scheme to obtain a triconnectivity labeling scheme using a reduction from [21]. There a labeling scheme for triconnectivity is given using labels of length bounded by $5 \log n$. By Lemmas 3.3, 3.4, and 3.6 in [21] and Theorem 3 we obtain the following improvement.

THEOREM 4. *For graphs with n nodes there is a triconnectivity labeling scheme that assigns unique labels with label length bounded by* $3\log n + O(\log\log n)$.

5. Upper bounds for sibling and connectivity labeling schemes. In this section we consider labeling schemes for sibling queries and connectivity queries in a forest. First we consider sibling queries. If two nodes in the same tree can be given the same label, we can label the nodes with labels of length $\lceil \log n \rceil$ as follows. Partition the nodes into groups such that two nodes are siblings if and only if they belong to the same group. This construction gives $g \leq n$ groups, which are numbered $1, 2, \ldots, g$. Nodes in the same group are given the same label, namely, the number of the group. Now, two nodes are siblings if and only if they have the same label.

THEOREM 5. *For trees with n nodes there is a sibling labeling scheme with label length bounded by* $\lceil \log n \rceil$.

Next we show how to assign unique labels for trees with maximum degree Δ. We group the nodes as above. We assign to each node v two numbers: a group number $g(v)$ to answer sibling queries as above, and an individual number $i(v)$ to make its label unique. Two nodes in the same group will be given the same group number. Assume we have g groups $g_1, g_2, \ldots, g_g$. Let $|g_i|$ be the number of nodes in g_i. Using a Huffman code [15] we give each node in group g_i, a group number of length $\log n - \log|g_i| + O(1)$. The individual numbers given to the nodes in group g_i are simply $1, 2, \ldots, |g_i|$, of length $\log|g_i| + O(1)$. In total we use $\log n + O(1)$ bits for the group and individual numbers; however, coding these two numbers as one label, we also need to be able to separate these two numbers given the label of a node. We use the first $O(\log\log\Delta)$ bits of the label to code the length of the individual number as follows. The individual number in a tree with maximum degree Δ is at most Δ and can be represented with at most $q = \log\Delta + O(1)$ bits. To represent the length of the individual number we need $O(\log q) = O(\log\log\Delta)$ bits. Now, we also need to represent the length of the bit string representing the length of the individual number, but this can be done simply by using an unary code of length $O(\log\log\Delta)$.

THEOREM 6. *For trees with n nodes and maximum degree Δ there is a sibling labeling scheme that assigns unique labels with label length bounded by* $\log n + O(\log\log\Delta)$.

Using the same observations, grouping connected nodes, we get the next theorem.

THEOREM 7. *For forests with n nodes there is a connectivity labeling scheme that assigns unique labels with label length bounded by* $\log n + O(\log\log n)$.

It is straightforward to compute the above labeling schemes in $O(n)$ time and answer queries in $O(1)$ time assuming standard binary operations on a RAM.

6. Lower bounds. In this section we present a lower bound technique and subsequently give lower bounds for ancestor, connectivity, sibling, 1-relationship, 2-restricted distance, and biconnectivity labeling schemes.

If v is an ancestor of w *or* w is an ancestor of v, we say that v and w are weak ancestors. A lower bound for a weak ancestor labeling scheme is clearly a lower bound for an ancestor labeling scheme. The lower bound presented in this paper is for weak ancestor labeling schemes.

We will use the following technique to show this lower bound. First we give a family of trees $\mathcal{F}_\mathcal{A}$ where each tree consists of cn nodes for a constant c. We then show that any labeling scheme (which may use nonunique labels) for weak ancestor queries needs to use $\Omega(n\log n)$ different labels for $\mathcal{F}_\mathcal{A}$. If m different labels are necessary, then the label length must be at least $\log m$. Since $\log(cn\log n) = \log n + \Omega(\log\log n)$,

for any constant c, we establish the lower bound. A similar construction is used for the other lower bounds.

In some cases, e.g., in [7], the goal is to minimize the average length of labels instead of the maximum. We note that, using the above technique, our lower bounds also hold for the average length of labels.

6.1. Lower bound technique. Let $\mathcal{S}$ be a set of elements and let $e : \mathcal{S} \to \mathcal{D}$ be a function labeling $\mathcal{S}$ with elements from some domain $\mathcal{D}$. We will assume $|\mathcal{S}| = nk$, where k is an integer $\leq \log n$ and n is a power of two. We define a partition P of $\mathcal{S}$ into k boxes each of n elements. The elements in the ith box, $1 \leq i \leq \log n$, denoted by B_i are partitioned into $n/2^i$ *groups* each of 2^i elements.

LEMMA 13. *Let S, e, and k be as described above. If there exists a partition P such that the following two properties hold, then $|\mathcal{D}| = \Omega(nk)$:*

(i) *for two different elements $s_1, s_2 \in \mathcal{S}$, if s_1 and s_2 belong to the same box, then $e(s_1) \neq e(s_2)$,*

(ii) *for elements $s_1, s_2, s_3, s_4 \in \mathcal{S}$, if s_1 and s_2 belong to two different groups in the same box, $e(s_1) = e(s_3)$ and $e(s_2) = e(s_4)$, then s_3 and s_4 belong to two different groups.*

Proof. We will say the function e associates labels with the elements from $\mathcal{S}$. The elements associated with the same label are called *neighbors*. In the following we give a strategy to choose a subset S' of elements from $\mathcal{S}$, guaranteeing that for all $s_1, s_2 \in S'$, where $s_1 \neq s_2$, s_1 and s_2 will not be neighbors. We call a strategy with such a guarantee a *safe* strategy. The number of labels needed by e for $\mathcal{S}$ will be at least the size of S' since $|\mathcal{D}| \geq |S'|$ when S' is chosen by a safe strategy. We say an element is a *marked* element if it is chosen to belong to S'. Hence, no two elements with the same label will be marked. If one or more elements from a group are marked we say the group is marked. For a box B we let $M(B)$ denote the number of marked groups belonging to the box.

We first mark elements from the box B_k and next for B_i in order of decreasing i. All elements in B_k will be marked. From the first property of Lemma 13 there are no neighbors in the same box and the marking is therefore safe. When marking elements from the remaining boxes B_i, $i < k$, we keep the invariant that $M(B_i) \leq n/2^{i+1}$. Hence, we will mark elements from at most half of the groups belonging to B_i.

Let $F(i)$ be the set of groups belonging to the boxes B_j, $j \geq i$, and let $M(F(i))$ be the number of marked groups belonging to $F(i)$. Since we keep the invariant that $M(B_i) \leq n/2^{i+1}$ for $i < k$, we have that for $i \leq k$, $M(F(i)) \leq n/2^k + \sum_{j=i}^{k-1} n/2^{j+1} = n/2^i$. Next, we describe how to mark elements from B_i, after marking elements from B_j, $j > i$. If a group in B_i includes an element with a marked neighbor in B_j, $j > i$, we say the group is *closed*. If a group is not closed it is *open*.

Let $s_1, s_2 \in B_i$ belong to two different groups. If s_1 has a marked neighbor s_3 and s_2 has a marked neighbor s_4, then by the second property of Lemma 13, s_3 and s_4 must belong to two different marked groups from $F(i+1)$. Hence, for each closed group in B_i we can associate a marked group from $F(i+1)$ which will not be associated to any other group in B_i. Since the number of groups in B_i is $n/2^i$ and we keep the invariant that $M(F(i+1)) \leq n/2^{i+1}$, at least $n/2^{i+1}$ of the groups in B_i will be open. Since the elements from the open groups do not have a marked neighbor and none of them are neighbors by the first property of Lemma 13 it is safe to mark all elements from the $n/2^{i+1}$ open groups of B_i. This way we maintain the invariant of marking elements from at most half of the groups in B_i, $i < k$. Summarizing, we

mark all elements in B_k and half the elements from the remaining $k-1$ boxes. In total we mark $\Omega(nk)$ elements. □

In the following sections we will define different families of graphs for which the nodes from these graphs can be partitioned such that the labeling obeys the properties given in Lemma 13.

6.2. Ancestor labeling schemes. To show a lower bound for a weak ancestor labeling scheme we give a family $\mathcal{F}_{\mathcal{A}}$ of $\log n$ trees $\{T_1, T_2, \ldots, T_{\log n}\}$, each of size $2n+1$. We show that for a subset $\mathcal{S}$ of the nodes from $\mathcal{F}_{\mathcal{A}}$, where $|\mathcal{S}| = n \log n$, there is a partition P of $\mathcal{S}$, such that any e must obey the two properties in Lemma 13. This implies that at least $\Omega(n \log n)$ labels are needed and will conclude our proof.

The tree T_i in $\mathcal{F}_{\mathcal{A}}$ consists of a root node with $n/2^i$ children. Each child v is the root of a path $\rho(v)$ of length 2^i. Furthermore, each node on these paths has a child which is a leaf not belonging to the path.

We have $|V(T_i)| = 2(n/2^i)2^i + 1 = 2n+1$. We let $\mathcal{S}$ be the subset of nodes from $\mathcal{F}_{\mathcal{A}}$ which belongs to a path $\rho(v)$, where v is a child of one of the root nodes in the family. Hence, $|\mathcal{S}| = n \log n$. Box B_i is the subset of nodes from $\mathcal{S}$ which belongs to the tree T_i. The nodes from box B_i are partitioned into groups such that two nodes from the same group belong to the same path. Next we show that the two properties from Lemma 13 must be fulfilled for any weak ancestor labeling scheme (e, d) in this partition.

Consider the first property. Let $s_1, s_2 \in B_i$, $s_1 \neq s_2$. If s_1 and s_2 are weak ancestors, choose s_2 to be the node closer to the root. On the other hand, if s_1 and s_2 are not weak ancestors, then choose s_2 arbitrarily. Let c be the leaf in T_i which is the child of s_2. Note that in both cases s_1 and c are not weak ancestors and therefore $d(e(s_1), e(c)) \neq d(e(s_2), e(c))$, which implies that $e(s_1) \neq e(s_2)$.

Next we consider the second property. Let $s_1, s_2, s_3, s_4 \in \mathcal{S}$, where s_1 and s_2 belong to two different groups in the same box. This implies that s_1 and s_2 are not weak ancestors. Hence, if $e(s_1) = e(s_3)$ and $e(s_2) = e(s_4)$, then s_3 and s_4 are not weak ancestors and therefore s_3 and s_4 must belong to different groups.

THEOREM 8. *A weak ancestor labeling scheme for trees with n nodes needs label of length* $\log n + \Omega(\log \log n)$.

6.3. Connectivity labeling schemes. In this section we consider the minimum label length required to answer connectivity queries in a forest if the labels assigned to the nodes must be unique. Let $\mathcal{F}_{\mathcal{C}}$ be the family of $\log n$ forests F_i, $1 \leq i \leq \log n$, where F_i consist of $2^{\log n - i}$ paths of length 2^i. We have $|V(F_i)| = n$. We let $\mathcal{S}$ be the nodes from $\mathcal{F}_{\mathcal{C}}$. Box B_i is the nodes from F_i. The nodes in B_i are partitioned into groups such that connected nodes are in the same group.

The first property from Lemma 13 follows trivially from our assumption that the labels assigned to a forest F_i are unique. Let $s_1, s_2, s_3, s_4 \in \mathcal{S}$. If s_1 and s_2 belong to two different groups from the same box B_i, s_1 and s_2 are not connected in F_i. If s_3 and s_4 are in the same group, s_3 and s_4 are connected in some forest, and $d(e(s_1), e(s_2))$ should therefore be different from $d(e(s_3), e(s_4))$, which cannot be the case if $e(s_1) = e(s_3)$ and $e(s_2) = e(s_4)$.

THEOREM 9. *A connectivity labeling scheme for forests with n nodes that assigns unique labels needs labels of length* $\log n + \Omega(\log \log n)$.

6.4. Sibling labeling schemes. In this section we consider the minimum label length required to answer sibling queries in a tree if the labels assigned to the nodes must be unique. We consider a forest of trees $\mathcal{F}_{\mathcal{S}}(k)$ of k trees T_i, $1 \leq i \leq k \leq \log n$. Let $B(j)$ be a complete balanced binary rooted tree with 2^j leaves and $2^{j+1} - 1$ nodes.

The tree T_i consists of a tree $B = B(\log n - i)$, where each leaf from B in T_i has 2^i children. These children are the set $\mathcal{S}$. The box B_i consists of the subset of nodes from $\mathcal{S}$ which comes from T_i. The nodes in box B_i are partitioned into groups such that two nodes which belong to the same group are siblings. The first property from Lemma 13 follows trivially from our assumption that the labels assigned to a tree are unique. Let $s_1, s_2, s_3, s_4 \in \mathcal{S}$. Since s_1 and s_2 does not belong to the same group, s_1 and s_2 are not siblings. If s_3 and s_4 belongs to the same group, s_3 and s_4 are siblings. Therefore $d(e(s_1), e(s_2))$ should be different from $d(e(s_3), e(s_4))$, which cannot be the case if $e(s_1) = e(s_3)$ and $e(s_2) = e(s_4)$. The maximum degree Δ of a tree in $\mathcal{F}_{\mathcal{S}}(k)$ is 2^k, and $|\mathcal{S}| = nk$, giving the next theorem.

THEOREM 10. *A sibling labeling scheme for trees with n nodes and maximum degree Δ that assigns unique labels needs labels of length $\log n + \Omega(\log\log\Delta)$.*

6.5. 1-relationship and 2-restricted distance labeling schemes. In this section we consider the minimum label length required to answer 1-relationship and 2-restricted distance queries in a tree. To show the bound for 1-relationship labeling schemes we show that a labeling scheme for answering both parent and sibling queries needs to use labels of length $\log + \Omega(\log\log n)$. Let $\mathcal{F}_{\mathcal{SP}}$ be the forest $\mathcal{F}_{\mathcal{S}}(\log n)$ to which we have added a child to each leaf in the forest $\mathcal{F}_{\mathcal{S}}(\log n)$. We let $\mathcal{S}$ be the same subset of nodes as in the previous section. Let s_1, s_2 belong to the same box, $s_1 \neq s_2$, and let c be the child of s_1. Since s_2 is not a parent to c, s_1 and s_2 must be assigned different labels. Hence, the first property of Lemma 13 is satisfied.

THEOREM 11. *A 1-relationship labeling for trees with n nodes needs labels of length $\log n + \Omega(\log\log n)$.*

For 2-restricted distance labeling schemes we use $\mathcal{F}_{\mathcal{SP}}$ and the same partition as above. Let s_1, s_2 belong to the same box, $s_1 \neq s_2$, and let c be the child of s_1. Since the distance from s_1 to c is 1 and the distance from s_2 to c is 3, s_1 and s_2 must be assigned different labels. Furthermore, the distance between two nodes in $\mathcal{S}$ is 2 if and only if they are siblings, and by the same observations as in the sibling labeling scheme the result follows.

THEOREM 12. *A 2-restricted distance labeling scheme for trees with n nodes needs labels of length $\log n + \Omega(\log\log n)$.*

6.6. Biconnectivity labeling schemes. In this section we consider the minimum label length required to answer biconnectivity queries in a graph. Let G_i be the graph consisting of 2^i disjoint cycles $C_i = \{c_1, \ldots, c_{2^i}\}$ each of length $n/2^i$. Furthermore, for each node $v \in V(C_i)$, G_i contain two nodes $v_1, v_2 \notin V(C_i)$ connected with each other and v. Let $\mathcal{F}_{\mathcal{B}}$ be the family $G_i, 1 \leq i \leq \log n - 2$, and let $\mathcal{S}$ be the set of nodes in $C_i, 1 \leq i \leq \log n - 2$. Then $|\mathcal{S}| = n(\log n - 2)$. The box B_i is the nodes in $\mathcal{S}$ from G_i, and two nodes are in the same group if they are biconnected. Note that cycles of length less than 3 are not biconnected and therefore the restriction $i \leq \log n - 2$ is important. Let $s_1, s_2 \in \mathcal{S}$ belong to the same box, $s_1 \neq s_2$ and let v_1 and v_2 be the nodes connected to s_1 but not on the cycle containing s_2. Since v_1 and v_2 are biconnected with s_1 but not s_2, $e(s_1) \neq e(s_2)$. Let $s_1, s_2, s_3, s_4 \in \mathcal{S}$, where s_1 and s_2 belong to different groups in the same box. This implies that s_1 and s_2 are not biconnected and if $e(s_1) = e(s_3)$ and $e(s_2) = e(s_4)$, s_3 and s_4 must also belong to different groups.

THEOREM 13. *A biconnectivity labeling scheme for graphs with n nodes needs labels of length $\log n + \Omega(\log\log n)$.*

Acknowledgments. We thank the anonymous reviewers for the very insightful and useful comments and Inge Li Gørtz for proofreading.

REFERENCES

[1] S. Abiteboul, H. Kaplan, and T. Milo, *Compact labeling schemes for ancestor queries*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 547–556.

[2] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, *Nearest common ancestors: A survey and a new distributed algorithm*, in Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architecture, 2002.

[3] S. Alstrup and T. Rauhe, *Improved labeling schemes for ancestor queries*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[4] S. Alstrup and T. Rauhe, *Small induced universal graphs and compact implicit graph representations*, in Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002.

[5] M. A. Breuer, *Coding vertexes of a graph*, IEEE Trans. Inform. Theory, 12 (1966), pp. 148–153.

[6] M. A. Breuer and J. Folkman, *An unexpected result on coding vertices of a graph*, J. Math. Anal. Appl., 20 (1967), pp. 583–600.

[7] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick, *Reachability and distance queries via 2-hop labels*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[8] E. Cohen, H. Kaplan, and T. Milo, *Labeling dynamic XML trees*, in Proceedings of the 21st Annual ACM Symposium on Principles of Database Systems, 2002.

[9] R. Diestel, *Graph Theory*, Springer-Verlag, New York, 2000.

[10] C. Gavoille, M. Katz, N. Katz, C. Paul, and D. Peleg, *Approximate distance labeling schemes*, in Proceedings of the 9th Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 2161, Springer-Verlag, New York, 2001, pp. 476–488.

[11] C. Gavoille and C. Paul, *Split decomposition and distance labeling: An optimal scheme for distance hereditary graphs*, in Proceedings of the 9th European Conference on Combinatorics, Graph Theory and Applications, 2001.

[12] C. Gavoille and D. Peleg, *Compact and localized distributed data structures*, Distributed Computing, 16 (2003), pp. 111–120.

[13] C. Gavoille, D. Peleg, S. Perennes, and R. Raz, *Distance labeling in graphs*, in Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms, 2001.

[14] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[15] D. A. Huffman, *A method for construction of minimum-redundancy codes*, in Proceedings of the Institute of Radio Engineers, 1952.

[16] S. Kannan, M. Naor, and S. Rudich, *Implicit representation of graphs*, in Proceedings of 20th Annual ACM Symposium on Theory of Computing, 1988.

[17] S. Kannan, M. Naor, and S. Rudich, *Implicit representation of graphs*, SIAM J. Discrete Math., 5 (1992), pp. 596–603.

[18] H. Kaplan and T. Milo, *Short and simple labels for small distances and other functions*, in Proceedings of the 7th Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 2125, Springer-Verlag, New York, 2001.

[19] H. Kaplan, T. Milo, and R. Shabo, *A comparison of labeling schemes for ancestor queries*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[20] M. Katz, N. Katz, and D. Peleg, *Distance labeling schemes for well-separated graph classes*, in Proceedings of the 17th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1170, Springer-Verlag, New York, 2000.

[21] M. Katz, N. A. Katz, A. Korman, and D. Peleg, *Labeling schemes for flow and connectivity*, in Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[22] D. Peleg, *Proximity-preserving labeling schemes and their applications*, in Graph-Theoretic Concepts in Computer Science, 25th International Workshop, Lecture Notes in Comput. Sci. 1665, Springer-Verlag, New York, 1999, pp. 30–41.

[23] N. Santoro and R. Khatib, *Labeling and implicit routing in networks*, Comput. J., 28 (1985), pp. 5–8.

[24] R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[25] M. Thorup and U. Zwick, *Approximate distance oracles*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, 2001, pp. 1–10.

[26] M. Thorup and U. Zwick, *Compact routing schemes*, in Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architecture, Vol. 13, 2001.

[27] A. K. Tsakalidis, *Maintaining order in a generalized linked list*, Acta Inform., 21 (1984), pp. 101–112.