Labeling Schemes for Small Distances in Trees

Stephen Alstrup* Philip Bille* 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.

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*, $l(v)$, 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 or not 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 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] shows 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 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 is 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.

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

*IT University of Copenhagen, Glentevej 67, DK-2400 Copenhagen NV, Denmark. Email: {stephen,beetle,theis}@it-c.dk

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

scheme for biconnectivity using $3\log n$ bits. We show, giving upper and lower bounds, that labels of length $\log n + \Theta(\log\log n)$ is 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 $3\log n + O(\log\log n)$.

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 non-trivial 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 the same label we say that the labels are unique. The above labeling schemes all produce unique labels, whereas the lower bounds also holds for labeling schemes that produce non-unique labels. However, the following bounds shows that there is a non-trivial complexity difference between labeling schemes assigning unique and non-unique labels.

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. If uniqueness is required, as in [16], we give upper and lower bounds showing that such a labeling scheme uses labels of length $\log n + \Theta(\log\log\Delta)$, for trees of maximum degree Δ. Extending the result for the sibling labeling scheme we give a labeling scheme supporting connectivity queries for forest using labels of length $\lceil\log n\rceil$ for forests with n nodes. Again, these labels are not unique and if this is required we show that such a labeling scheme requires labels of length $\log n + \Theta(\log\log n)$.

1.1 Related work Adjacency labeling schemes, were introduced by Breuer and Folkman [5, 6] and efficient labeling schemes were consider 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 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].

1.2 Outline In Section 2 we give some preliminaries and in Section 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 maximum degree of T is the maximum number of children of any node $v \in V(T)$. 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 node $A(v,1)$ 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 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 $a = a_0 a_1 \ldots a_{n-1}$, where $a_i \in \{0,1\}$, $0 \leq i \leq n-1$. For $0 \leq j \leq n-1$ the sequences $a_0, \ldots, a_{j-1}$ and $a_{n-j}, \ldots, a_{n-1}$ 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 $a_0 \ldots a_{r-1}$, 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 $\{i, \ldots, j\}$.

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 $f_1, \ldots, f_m$ ($f_i : V(G) \times V(G) \to X_i$) is a tuple $(e, d_1, \ldots, d_m)$ 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, then 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 for any $G \in \mathcal{F}$ there is an encoder which assigns labels to all nodes in $V(G)$ 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 of this 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 techniques presented below are similar to the ones used in [4]. Here a labeling scheme supporting parent queries is given with labels of length bounded by $\log n + O(\log \log n)$. This result is similar to Lemma 3.7. In [4], however, only the parent query is considered and it is shown that the label length can be reduced to $\log n + O(\log^* n)$. In this section we instead generalize the result to relationship queries. 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 $L(w)$ 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 3.1. ([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 an internal node is the heavy node. The light children may 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 $v \in L(w)$. We have the following relation between significant ancestors and the preorder numbering.

LEMMA 3.2. *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 $\omega = \lceil \log n \rceil$ and $f(v) = \lfloor \log \text{lightsize}(v) \rfloor$. We define the *significant preorder number*, $\text{spre}(v)$, as a number in $[\text{pre}(v), \text{pre}(v) + \text{lightsize}(v) - 1]$ where all the $f(v)$ least significant bits are 0: If $\text{pre}(v) \bmod 2^{f(v)} = 0$ then $\text{spre}(v) = \text{pre}(v)$. Otherwise $\text{spre}(v) = \text{pre}(v) - (\text{pre}(v) \bmod 2^{f(v)}) + 2^{f(v)}$. The following lemma states the relations we need between the preorder and significant preorder numbers.

LEMMA 3.3. *For all nodes v and w the following holds:*

(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* $\text{spre}(w) < \text{spre}(v)$.

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 3.2 $\text{pre}(v) \notin [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ 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) If $\text{pre}(w) < \text{pre}(v)$ and $\text{lightdepth}(v) = \text{lightdepth}(w)$, then $v \notin L(w)$. By Lemma 3.2, $\text{pre}(v) \notin [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$ 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 3.3(ii) a 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 3.4. *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 3.2, $\text{pre}(v) \in [\text{pre}(w), \text{pre}(w) + \text{lightsize}(w) - 1]$. 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 3.1 $\text{lightdepth}(v) \leq \log n + O(1)$, 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 3.4 that $\text{spre}(v)$ and $\text{spre}(\text{parent}(v))$ can also be represented using $\log\log n + O(1)$ bits. By Lemma 3.3(ii), $\text{lightdepth}(v)$ together with $\text{spre}(v)$ uniquely identifies the node v. This immediately implies the following:

LEMMA 3.5. *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 3.6. *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. For internal nodes, define $\text{diff\_heavy}(v) = \text{spre}(\text{heavy}(v)) - \text{spre}(v)$ and leave it undefined for leaves. We will use the discrete logarithm of $\text{diff\_parent}(v)$ and $\text{diff\_heavy}(v)$ to test for parentship 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 3.7. *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 3.3, 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 3.3(iii) that $\text{pre}(w) < \text{pre}(v)$. Then, since $\text{lightdepth}(w) = \text{lightdepth}(\text{heavy}(w)) = \text{lightdepth}(\text{parent}(v)) = \text{lightdepth}(v)$ and $w \neq \text{parent}(v)$, this implies that $\text{pre}(w) < \text{pre}(\text{heavy}(w)) \leq \text{pre}(\text{parent}(v)) < \text{pre}(v)$. By Lemma 3.3(ii) and (iii), also $\text{spre}(\text{heavy}(w)) \leq \text{spre}(\text{parent}(v))$ and therefore $\text{spre}(v) - \text{spre}(w) \geq (\text{spre}(\text{heavy}(w)) - \text{spre}(w)) + (\text{spre}(v) - \text{spre}(\text{parent}(w))) = \text{diff\_heavy}(w) + \text{diff\_parent}(v)$. 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 $\text{spre}(v) - \text{spre}(w) \geq \text{diff\_heavy}(w) + \text{diff\_parent}(v) \geq 2 \cdot 2^{\lfloor \log \text{diff\_parent}(v) \rfloor} = 2 \cdot 2^{\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor} > \text{spre}(v) - \text{spre}(w)$. □

By the same reasoning we obtain the following:

LEMMA 3.8. *For a heavy node v and light node w, v and 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 $\text{diff\_heavy}(\text{parent}(v))$. 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 prefix the label with the binary representation of the length of the field containing $\text{pre}(v)$. Since the length of this field is $\lceil \log n \rceil$ this can be done using $\log \log n + O(1)$ bits. Now, we also need to store the length of the bit string representing the length of $\text{pre}(v)$, but this can be done using a unary code of length at most $\log \log n + O(1)$. In total the label length will be bounded by $\log n + O(\log \log n)$. By uniqueness of the labels and Lemmas 3.5 – 3.8 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:

THEOREM 3.1. *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 $\text{diff\_h\_des}(v,m) = \text{spre}(u) - \text{spre}(v)$ and if there is no such node u let $\text{diff\_h\_des}(v,m) = 2n$, *i.e.,* the discrete logarithm of $2n$ will be $\lfloor \log n \rfloor + 1$ indicating that this is not an actual difference. Similarly, define $\text{diff\_h\_anc}(v,m)$ 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 3.7:

LEMMA 3.9. *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* $\text{spre}(w) < \text{spre}(v)$, $\text{lightdepth}(v) = \text{lightdepth}(w)$, $\lfloor \log \text{spre}(v) - \text{spre}(w) \rfloor = \lfloor \log \text{diff\_h\_anc}(v,m) \rfloor = \lfloor \log \text{diff\_h\_des}(w,m) \rfloor$ *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 $x = w$ (or $y = v$) it is straightforward to check that the conditions are satisfied. Conversely, assume that the conditions are satisfied and $x \neq w$. Since $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_h\_anc}(v,m) \rfloor = \lfloor \log \text{diff\_h\_des}(w,m) \rfloor$ both x and y exists and are on the same heavy paths as v and w respectively. As in the proof of Lemma 3.7, since $\text{lightdepth}(w) = \text{lightdepth}(y) = \text{lightdepth}(x) = \text{lightdepth}(v)$, $\text{spre}(w) < \text{spre}(v)$, $\text{index}(v) \bmod m = \text{index}(w) \bmod m$ and $x \neq w$, it follows that $\text{spre}(y) \leq \text{spre}(x)$. Then $\text{spre}(v) - \text{spre}(w) \geq \text{diff\_h\_des}(w,m) + \text{diff\_h\_anc}(v,m)$. By the identities $\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor = \lfloor \log \text{diff\_h\_anc}(v,m) \rfloor = \lfloor \log \text{diff\_h\_des}(w,m) \rfloor$ this leads to the contradiction $\text{spre}(v) - \text{spre}(w) \geq \text{diff\_h\_des}(w,m) + \text{diff\_h\_anc}(v,m) \geq 2 \cdot 2^{\lfloor \log \text{diff\_h\_anc}(v,m) \rfloor} = 2 \cdot 2^{\lfloor \log(\text{spre}(v) - \text{spre}(w)) \rfloor} > \text{spre}(v) - \text{spre}(w)$. □

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.

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 pre(v) and 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 spre(w), dist(v,w) and a single bit, called the *apex bit*, indicating whether or not the distance to apex(w) is at most k. If this is so we store dist$(w,\text{apex}(w))$ and otherwise leave this field undefined. Furthermore, the ith entry also represents, for $1 \leq m \leq k$, diff_heavy(w,m), diff_parent(w,m) and index(w) mod 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 $O(k^2(\log\log n + \log k))$. 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 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.

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 3.10. *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* lightdepth(v') = lightdepth$(A(v,k_1))$ *and if $v' \neq v$ let v'' be the significant ancestor of v of light depth* lightdepth$(A(v,k_1)+1$. *Otherwise let $v'' = v$. Similarly, define w' and w'' for w. Then, v and w are (k_1,k_2)-related if and only if one of the following disjoint conditions are satisfied.*

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

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

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

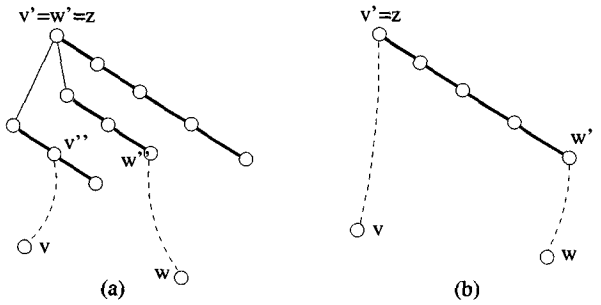

Figure 1: Cases for Lemma 3.10. (a) case (i). (b) case (ii).

Proof. The situation is illustrated in figure 1. Let $z = \text{nca}(v,w)$. If one of the conditions are 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 not both of k_1 and k_2 are zero, v'' and w'' must be on different heavy paths, since 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 that not both of k_1 and k_2 are zero. We will show how to test each of the conditions in Lemma 3.10 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''. So assume that the values stored at these entries of the tables are available. Using Lemma 3.3(ii) we can check if $v' = w'$. The distances dist(v,v') and 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 3.9. What remains is to describe how to test if v'' and w'' are on different heavy paths. Since the distances dist$(v'',\text{apex}(v''))$ and 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 $|\text{dist}(v'',\text{apex}(v'')) - \text{dist}(w'',\text{apex}(w''))|$, and we can thus apply Lemma 3.10 to test the condition. In summary we have shown:

THEOREM 3.2. *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 assigns 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 maximally 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 3 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 contains v and has 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:

LEMMA 4.1. *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 4.2. *Two nodes v and w are biconnected in G if and only if, in the block forest of rooted trees B_r, v and w are siblings, v is the grandparent of w or vice versa.*

Proof. If v and w are biconnected in G then they are contained in the same block with at least tree 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 tree nodes and are thus biconnected. □

To test the conditions in Lemma 4.2 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 4.2:

THEOREM 4.1. *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]. Here 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 4.1 we obtain the following improvement.

THEOREM 4.2. *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 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 label 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 $1, 2 \cdots g$. 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 5.1. *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 $g_1, g_2 \cdots g_g$. 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 $1, 2, \cdots |g_i|$, 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 5.2. *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.

THEOREM 5.3. *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.

A test for whether or not v is an ancestor of w we denote as an ancestor test, whereas a test for either v is ancestor of w or vice versa is called a weak ancestor test. A lower bound for weak ancestor tests is clearly a lower bound for ancestor tests. The lower bound presented in this paper is for weak ancestor tests. We will use the following technique to show this lower bound: First we give a family of trees $\mathcal{F}_A$ where each tree consist of cn nodes for a constant c. We then show that any labeling scheme (which may use non-unique labels) for weak ancestor queries need to use $\Omega(n \log n)$ different labels for $\mathcal{F}_A$. 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 holds for the average length of labels.

6.1 Lower bound technique Let $\mathcal{S}$ be a set of elements and let $l : \mathcal{S} \to \mathcal{D}$ 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 B_i are partitioned into $n/2^i$ *groups* each of 2^i elements.

LEMMA 6.1. *Let S, l and k be as described above. If there exists a partition P such that the two properties hold:*

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

(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, $l(s_1) = l(s_3)$ and $l(s_2) = l(s_4)$, then s_3 and s_4 belong to two different groups.

then $|\mathcal{D}| = \Omega(nk)$.

Proof. We will say the function l associate labels to the elements from $\mathcal{S}$. The elements associated with the same label we denote as *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 for a *safe* strategy. The number of labels needed by l for $\mathcal{S}$ will be at least the size of S' since $|\mathcal{D}| \geq |S'|$ when choosing S' by a safe strategy. An element chosen to belong to S', we say is a *marked* element. 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 6.1 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 at most mark elements from 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 $i \leq k$, $M(F(i)) \leq n/2^k + \sum_{j=i}^{j=k-1} n/2^{i+1} = n/2^i$. 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 denote the group as *closed*. If a group not is closed we denote it as *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 6.1, 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 groups 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 does not have a marked neighbor and none of them are neighbors by the first property of Lemma 6.1 it is safe marking all elements from $n/2^{i+1}$ groups of B_i. Doing this we keep the invariant, at most marking elements from half of the groups in B_i, $i < k$. Summarizing, we mark all elements in B_k, and half of 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 6.1.

6.2 Ancestor labeling schemes To show a lower bound for an ancestor labeling l, we give a family $\mathcal{F}_\mathcal{A}$ of $\log n$ trees $\{T_1, T_2, \cdots, T_{\log n}\}$, 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 l, must obey the two properties in Lemma 6.1. 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 have 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 6.1 must be fulfilled for any labeling l in this partition.

Consider the first property. Let $s_1, s_2 \in B_i$, $s_1 \neq s_2$. If s_1 is an ancestor to s_2, s_2 cannot be an ancestor of s_1. Assume without loss of generality that s_1 is not an ancestor of s_2 and let c be the leaf in T_i which is the child of s_2. Since s_1 is not an ancestor of s_2, s_1 cannot be ancestor of c. Therefore the decoder for the ancestor query, d_a, must satisfy that $d_a(l(s_1), l(c)) \neq d_a(l(s_2), l(c))$, which implies that $l(s_1) \neq l(s_2)$.

Next we consider the second property. Let $s_1, s_2, s_3, s_4 \in \mathcal{S}$, where s_1 and s_2 belongs to two different groups in the same box. This implies that there is no ancestor relation between s_1 and s_2. Thus if $l(s_1) = l(s_3)$ and $l(s_2) = l(s_4)$ there cannot be an ancestor relation between s_3 and s_4 and therefore s_3 and s_4 must belongs to different groups. Hence, we have shown the following theorem:

THEOREM 6.1. *An ancestor labeling scheme for trees with n nodes needs label of length* $\log n + \Omega(\log \log n)$.

Using the same approach we obtain the following lower bounds. The proofs can be found in the full version of this paper.

THEOREM 6.2. *A connectivity labeling scheme for forests with n nodes, that assign unique labels, needs labels of length* $\log n + \Omega(\log \log n)$.

THEOREM 6.3. *A sibling labeling scheme for trees with n nodes and maximum degree Δ, that assigns unique labels, needs labels of length* $\log n + \Omega(\log \log \Delta)$.

THEOREM 6.4. *A 1-relationship labeling for trees with n nodes needs labels of length* $\log n + \Omega(\log \log n)$.

THEOREM 6.5. *A 2-restricted distance labeling scheme for trees with n nodes needs labels of length* $\log n + \Omega(\log \log n)$.

THEOREM 6.6. *A biconnectivity labeling scheme for graphs with n nodes needs labels of length* $\log n + \Omega(\log \log n)$.

References

[1] 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)*, pages 547–556, 2001.

[2] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors : A survey and a new distributed algorithm. In *In Proceedings of the fourteenth annual ACM Symposium on Parallel Algorithms and Architecture (SPAA)*, 2002.

[3] S. Alstrup and T. Rauhe. Improved labeling schemes for ancestor queries. In *Prooceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.

[4] S. Alstrup and T. Rauhe. Small induced universal graphs and compact implicit graph representations. In *Proceedings of the fourtythird annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

[5] M. A. Breuer. Coding vertexes of a graph. *IEEE Transactions on Information Theory*, IT–12:148–153, 1966.

[6] M. A. Breuer and J. Folkman. An unexpected result on coding vertices of a graph. *Journal of Mathematical Analysis and Applications*, 20:583–600, 1967.

[7] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.

[8] E. Cohen, H. Kaplan, and T. Milo. Labeling dynamic xml trees. In *Prooceedings of the twentyfirst annual ACM Symposium on Principles of Database Systems (PODS)*, 2002.

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

[10] C. Gavoille, M. Katz, N. Katz, C. Paul, and D. Peleg. Approximate distance labeling schemes. In *Proceedings of the ninth annual European Symposium on Algorithms (ESA)*, volume 2161 of *LNCS*, pages 476–488. Springer Verlag, 2001.

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

[12] C. Gavoille and D. Peleg. Compact and localized distributed data structures. Technical Report RR-1261-01, Laboratoire Bordelais de Recherce en Informatique, 2001.

[13] C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. In *Proceedings of the twelfth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2001.

[14] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal of Computing*, 13(2):338–355, 1984.

[15] D. A. Huffman. A methode for construction of minimum-redundancy codes. *Proceedings of the IRE*, 1952.

[16] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. In *Proceedings of twentieth annual ACM-SIAM Symposium On Theory of Computing (STOC)*, 1988.

[17] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 1992. Preliminary version appeared in STOC'88.

[18] H. Kaplan and T. Milo. Short and simple labels for small distances and other functions. In *Proceeding of the seventh Workshop on Algorithms and Data Structures*, LNCS, 2001.

[19] 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.

[20] M. Katz, N. Katz, and D. Peleg. Distance labeling schemes for well-seperated graph classes. In *Proceedings of the seventeenth Symposium on Theorectical Aspects of Computer Science (STACS)*, volume 1170 of *LNCS*. Springer Verlag, 2000.

[21] M. Katz, N. A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. In *Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2002.

[22] D. Peleg. Proximity-preserving labeling schemes and their applications. In *Graph-Theoretic Concepts in Computer Science, twentyfifth international workshop*, volume 1665 of *LNCS*, pages 30–41. Springer Verlag, 1999.

[23] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The computer Journal*, 28:5–8, 1985.

[24] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[25] M. Thorup and U. Zwick. Approximate distance oracles. In *In Proceedings of the thirteenth annual ACM-SIAM Symposium on Theory of Computing (STOC)*, pages 1–10, 2001.

[26] M. Thorup and U. Zwick. Compact routing schemes. In *In Proceedings of the thirteenth annual ACM Symposium on Parallel Algorithms and Architecture (SPAA)*, volume 13, 2001.

[27] A. K. Tsakalidis. Maintaining order in a generalized linked list. *Acta Informatica*, 21(1):101–112, 1984.