Near-optimal labeling schemes for nearest common ancestors

Stephen Alstrup[*] Esben Bistrup Halvorsen[†]

Kasper Green Larsen[‡]

Abstract

We consider NCA labeling schemes: given a rooted tree T, label the nodes of T with binary strings such that, given the labels of any two nodes, one can determine, by looking only at the labels, the label of their nearest common ancestor.

For trees with n nodes we present upper and lower bounds establishing that labels of size $(2 \pm \epsilon)\log n$, $\epsilon < 1$ are both sufficient and necessary.[1]

Alstrup, Bille, and Rauhe (SIDMA'05) showed that ancestor and NCA labeling schemes have labels of size $\log n + \Omega(\log\log n)$. Our lower bound increases this to $\log n + \Omega(\log n)$ for NCA labeling schemes. Since Fraigniaud and Korman (STOC'10) established that labels in ancestor labeling schemes have size $\log n + \Theta(\log\log n)$, our new lower bound separates ancestor and NCA labeling schemes. Our upper bound improves the $10\log n$ upper bound by Alstrup, Gavoille, Kaplan and Rauhe (TOCS'04), and our theoretical result even outperforms some recent experimental studies by Fischer (ESA'09) where variants of the same NCA labeling scheme are shown to all have labels of size approximately $8\log n$.

1 Introduction

A *labeling scheme* assigns a *label*, which is a binary string, to each node of a tree such that, given only the labels of two nodes, one can compute some predefined function of the two nodes. The main objective is to minimize the *maximum label length*: that is, the maximum number of bits used in a label.

With labeling schemes it is possible, for instance, to avoid costly access to large, global tables, to compute locally in distributed settings, and to have storage used for names/labels be informative. These properties are used in XML search engines [2], network routing and distributed algorithms [57, 29, 22, 24, 29, 30], graph representations [40] and other areas. An extensive survey of labeling schemes can be found in [35].

A nearest common ancestor (NCA) labeling scheme labels the nodes such that, for any two nodes, their labels alone are sufficient to determine the label of their NCA. Labeling schemes can be found, for instance, for distance, ancestor, NCA, connectivity, parent and sibling [36, 44, 51, 7, 40, 57, 8, 41, 15, 16, 45, 55], and have also been analyzed for dynamic trees [20]. NCA labeling schemes are used, among other things, to compute minimum spanning trees in a distributed setting [50, 28, 13].

[*] Department of Computer Science, University of Copenhagen, Denmark, s.alstrup@diku.dk.

[†] Department of Computer Science, University of Copenhagen, Denmark, esbenbh@diku.dk.

[‡] MADALGO - Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation, Department of Computer Science, Aarhus University, Denmark, larsen@cs.au.dk.

[1] All logarithms in this paper are in base 2.

Our main result establishes that labels of size $(2 \pm \epsilon)\log n$, $\epsilon < 1$ are both necessary and sufficient for NCA labeling schemes for trees with n nodes. More precisely, we show that label sizes are lower bounded by $1.008\log n - O(1)$ and upper bounded by $2.772\log n + O(1)$.

Since our lower bound is $\log n + \Omega(\log n)$, this establishes an exponential separation (on the nontrivial, additive term) between NCA labeling and the closely related problem of ancestor labeling which can be solved optimally with labels of size $\log n + \Theta(\log\log n)$ [32, 6]. (An ancestor labeling scheme labels the nodes in a tree such that, for any two nodes, their labels alone are sufficient to determine whether the first node is an ancestor of the second.) The upper bound of $\log n + \mathrm{O}(\log\log n)$ for ancestor [32] is the latest result in a sequence [2, 41, 42, 8, 1, 31] of improvements from the trivial $2\log n$ bound [58].

Our upper bound improves the $10\log n$ label size of [7]. In addition to the NCA labeling scheme used to establish our upper bound, we present another scheme with labels of size $3\log n$ which on the RAM uses only linear time for preprocessing and constant time for answering queries, meaning that it may be an efficient solution for large trees compared to traditional non-labeling scheme algorithms [38].

NCAs, also known as *least common ancestors* or *lowest common ancestors* (LCAs), have been studied extensively over the last several decades in many variations; see, for example, [48, 3, 5, 56, 21, 10, 53, 11, 33, 56, 12]. A linear time algorithm to preprocess a tree such that subsequent NCA queries can be answered in constant time is described in [38]. NCAs have numerous applications for graphs [34, 43, 23, 5], strings [37, 25], planarity testing [59], geometric problems [18, 33], evolutionary trees [26], bounded tree-width algorithms [19] and more. A survey on NCAs with variations and application can be found in [7].

A $\log n + \mathrm{O}(\log^* n)$ adjacency labeling scheme is presented in [9], and adjacency labeling schemes of $\log n + \mathrm{O}(1)$ are presented in [14] for the special cases of binary trees and caterpillars. We present NCA labeling schemes with labels of size $2.585\log n + O(1)$ and $\log n + \log\log n + O(1)$ for binary trees and caterpillars, respectively. Our lower bound holds for any family of trees that includes all trees of height $O(\log n)$ in which all nodes either have 2 or 3 children.

1.1 Variations and related work.

The NCA labeling scheme in [7] is presented as an $\mathrm{O}(\log n)$ result, but it is easy to see that the construction gives labels of worst-case size $10\log n$. The algorithm uses a decomposition of the tree, where each component is assigned a sub-label, and a label for a node is a combination of sub-labels. Fischer [27] ran a series of experiments using various techniques for sub-labels [7, 49, 39] and achieved experimentally that worst-case label sizes are approximately $8\log n$.

Peleg [52] has established labels of size $\Theta(\log^2 n)$ for NCA labeling schemes in which NCA queries have to return a *predefined label* of $\mathrm{O}(\log n)$ bits. Experimental studies of this variation can be found in [17]. In [13] the results from [7] are extended to predefined labels of length k. We have included a corollary that shows that such an extension can be achieved by adding $k\log n$ bits to the labels.

In [46] a model (1-query) is studied where one, in addition to the label of the input nodes, can access the label of one additional node. With this extra information, using the result from [7] for NCA labeling, they present a series of results for NCA and distance. As our approach improves the label length from [7], we also improve some of the label lengths in [46].

Sometimes various computability requirements are imposed on the labeling scheme: in [45] a query should be computable in polynomial time; in [2] in constant time on the RAM; and in [40] in polynomial time on a Turing machine. We use the same approach as in [7], but with a different kind of sub-labels and with different encodings for lists of strings, and it is only the $2.772 \log n + O(1)$ labeling scheme for trees that we do not show how to implement efficiently.

2 Preliminaries

The *size* or *length* of a binary string $s = s_1 \cdots s_k$ is the number of bits $|s| = k$ in it. The concatenation of two strings s and t is denoted $s \cdot t$.

Let T be a rooted tree with root r. The *depth* of a node v, denoted $\mathrm{depth}(v)$, is the length of the unique path from r to v. If a node u lies on the path from r to a node v, then u is an *ancestor* of v and v is a *descendant* of u. If, in addition, $\mathrm{depth}(v) = \mathrm{depth}(u) + 1$ so that uv is an edge in the tree, then u is the unique *parent* of v, denoted $\mathrm{parent}(v)$, and v is a *child* of u. A *binary tree* is a rooted tree in which any node has at most two children. A *common ancestor* of two nodes v and w is a node that is an ancestor of both v and w, and their *nearest common ancestor* (NCA), denoted $\mathrm{nca}(v, w)$, is the unique common ancestor with maximum depth. The descendants of v form an induced subtree T_v with v as root. The *size* of v, denoted $\mathrm{size}(v)$, is the number of nodes in T_v.

Let $\mathcal{T}$ be a family of rooted trees. An *NCA labeling scheme for* $\mathcal{T}$ consists of an *encoder* and a *decoder*. The encoder is an algorithm that accepts any tree T from $\mathcal{T}$ as input and produces a *label* $l(v)$, which is a binary string, for every node v in T. The decoder is an algorithm that takes two labels $l(v)$ and $l(w)$ as input and produces the label $l(\mathrm{nca}(v, w))$ as output. Note that encoder knows the entire tree when producing labels for nodes, whereas the decoder knows nothing about v, w or the tree from which they come, although it does know that they come from the *same* tree and that this tree belongs to $\mathcal{T}$. The worst-case *label size* is the maximum size of a label produced by the encoder from any node in any tree in $\mathcal{T}$.

3 Lower bound

This section introduces a class of integer sequences, 3-2 sequences, and an associated class of trees, 3-2 trees[2], so that two 3-2 trees that have many labels in common when labeled with an NCA labeling scheme correspond to two 3-2 sequences that are "close" in the sense of a metric known as Levenshtein distance. By considering a subset of 3-2 sequences that are pairwise distant in this metric, the corresponding set of 3-2 trees cannot have very many labels in common, which leads to a lower bound on the total number of labels and hence on the worst-case label size.

3.1 Levenshtein distance and 3-2 sequences.

The *Levenshtein distance* [47], or *edit distance*, between two sequences x and y is defined as the number $\mathrm{lev}(x, y)$ of single-character edits (insertion, deletion and substitution) required to transform x into y. A *3-2 sequence* of length $2k$ is an integer sequence $x = (x_1, \ldots, x_{2k})$ with exactly k 2s and k 3s.

Lemma 3.1. *For any h, k with $2 \le h \le k$ and k an integer with $k \ge 90$, there exists a set Σ of 3-2 sequences of length $2k$ with $|\Sigma| \ge 2^{1.95k}/(16k/h)^{3h}$ and $\mathrm{lev}(x, y) > h$ for all $x, y \in \Sigma$.*

Proof. Since $\mathrm{lev}(x, y) > h$ is equivalent to $\mathrm{lev}(x, y) > \lfloor h \rfloor$ and $2^{1.95k}/(16k/h)^{3h} \le 2^{1.95k}/(16k/\lfloor h \rfloor)^{3\lfloor h \rfloor}$, we can safely assume that h is an integer.

[2] The related "2-3 trees" [4] have a slightly different definition, which is why we use a different terminology here.

Now, let x be an arbitrary 3-2 sequence of length $2k$, and consider the number of 3-2 sequences y of length $2k$ with $\mathrm{lev}(x, y) \leq h$. We can transform x into y by performing r deletions followed by s substitutions followed by t insertions, where $r + s + t \leq h$. This leads to the following upper bound on the number of y's:

$$\sum_{r=0}^{h} \binom{2k}{r} \sum_{s=0}^{h-r} \binom{2k-r}{s} \sum_{t=0}^{h-r-s} \binom{2k-r-s+t}{t} 2^t \leq (h+1)^3 \binom{2k}{h}^2 \binom{3k}{h} 2^h.$$

Using Stirling's approximation [54] and the fact that $(h + 1)^3 \leq 8^h$ for all $h \geq 2$, it follows that this is upper bounded by

$$8^h(2ke/h)^{2h}(3ke/h)^h 2^h = 3^h(4ke/h)^{3h} \leq (16k/h)^{3h}.$$

We now construct Σ as follows. Let Σ' denote the set of 3-2 sequences of length $2k$, and note that $|\Sigma'| = \binom{2k}{k}$. Pick an arbitrary 3-2 sequence x from Σ', add it to Σ and remove all strings y from Σ' with $\mathrm{lev}(x, y) \leq h$. Continue by picking one of the remaining strings from Σ', add it to Σ and remove all strings from Σ' within distance h. When we run out of strings in Σ' we will, according to the previous calculation and Stirling's approximation [54], have

$$|\Sigma| \geq \frac{\binom{2k}{k}}{(16k/h)^{3h}} \geq \frac{2^{2k-1}}{k^{1/2}(16k/h)^{3h}} \geq \frac{2^{1.95k}}{(16k/h)^{3h}},$$

where the last inequality follows from the fact that $2^{0.05k-1} \geq k^{1/2}$ whenever $k \geq 90$. □

3.2 3-2 trees and a lower bound.

Given a 3-2 sequence $x = (x_1, \ldots, x_{2k})$ of length $2k$, we can create an associated tree of depth $2k$ where all nodes at depth $i - 1$ have exactly x_i children, and all nodes at depth $2k$ are leaves. We denote this tree the *3-2 tree associated with* x. The number of nodes at depth i in the 3-2 tree associated with x is $x_1 \cdots x_i$; in particular, the number of leaves is $x_1 \cdots x_{2k} = 6^k$. The number of nodes in total is upper bounded by $2 \cdot 6^k$.

Consider the set of labels produced by an NCA labeling scheme for the nodes in a tree. Given a subset S of these labels, let S' denote the set of labels which can be generated from S by the labeling scheme: thus, S' contains the labels in S as well as the labels for the NCAs of all pairs of nodes labeled with labels from S. The labels in S' can be organized as a rooted tree according to their ancestry relations, which can be determined directly from the labels using the decoder of the labeling scheme and without consulting the original tree. The tree produced in this way is denoted T^S and is uniquely determined from S. Note that, if all the nodes in S are leaves, then all internal nodes in T^S must have been obtained as the NCA of two leaves, and hence must have at least two children.

Now, given a tree T^S induced by a subset S of labels assigned to the leaves of a tree T by an NCA labeling scheme, we can create an integer sequence, $I(S)$, as follows. Start at the root of T^S, and let the first integer be the number of children of the root. Then recurse to a child v for which the subtree T^S_v contains a maximum number of leaves, and let the second integer be the number of children of this child. Continue this until a leaf is reached (without writing down the last 0). Note that, if T is a 3-2 tree of depth $2k$, the produced sequence $I(S)$ will have length at most $2k$ and will contain only 2s and 3s.

Lemma 3.2. *Let T be a 3-2 tree associated with the 3-2 sequence $x = (x_1, \ldots, x_{2k})$. Let S be a set of m labels assigned to the leaves of T by an NCA labeling scheme. Then* $\mathrm{lev}(x, I(S)) \leq \log_{3/2}(6^k/m)$.

Proof. We describe a way to transform x into $I(S)$. Start at the root of T, and let i be the depth in T containing the node v whose label $l(v)$ is the root in T^S. Delete all entries $x_1, \ldots, x_{i-1}$ from x and compare the number of children of $l(v)$ in T^S to x_i. If the numbers are the same, leave x_i be; if not, we must have that $x_i = 3$ and that the number of children of $l(v)$ is 2, so replace x_i by 2. Then recurse to a child w of v in T for which the corresponding subtree in T^S contains a maximum number of leaves, and repeat the process with T_w, the corresponding subtree of T^S and the remaining elements $x_{i+1}, \ldots, x_k$.

Clearly, this transforms x into $I(S)$ using only deletions and substitutions, where all substitutions replace a 3 by a 2. Each of these edits modify the maximum possible number of leaves in T^S compared to T with a factor of either $1/2$ or $2/3$. It follows that the number m of leaves in T^S satisfies $m \leq 6^k \cdot (2/3)^{\operatorname{lev}(x, I(S))}$, which implies $\operatorname{lev}(x, I(S)) \leq \log_{3/2}(6^k/m)$ as desired. □

We now present our main lower bound result. The result is formulated for a family $\mathcal{T}$ that is large enough to contain all 3-2 trees with N nodes; in particular, it holds for the family of all rooted trees with at most N nodes.

Theorem 3.3. *If $\mathcal{T}$ is a family of trees that contains all 3-2 trees with up to $N \geq 2 \cdot 3^{240}$ nodes, then any NCA labeling scheme for $\mathcal{T}$ has a worst-case label size of at least $1.008 \log N - 318$.*

Proof. Let $k = 120\lfloor \frac{1}{120} \log_6(N/2) \rfloor$ be $\log_6(N/2)$ rounded down to the nearest multiple of 120, and let $n = 6^k \leq N/2$. Further, set $m = n^{119/120}$ and $h = 2\log_{3/2}(n/m)$. Note that n, m and $n/m = n^{1/120}$ are all integers. Observe also that $n > (N/2)/6^{120} \geq (3/2)^{120}$ and thereby that $h \geq 2$. Finally, observe that $h = \frac{1}{60} k \log_{3/2} 6 \leq k$ and that $k \geq 120$.

According to Lemma 3.1, there exists a set Σ of 3-2 sequences of length $2k$ with $|\Sigma| \geq 2^{1.95k}/(16k/h)^{3h}$ and $\operatorname{lev}(x, y) > h$ for all $x, y \in \Sigma$. The set Σ defines a set of $|\Sigma|$ associated 3-2 trees with n leaves and at most $2n \leq N$ nodes. In particular, all the associated trees belong to $\mathcal{T}$. We can estimate the number of elements in Σ as follows:

$$\begin{aligned}
|\Sigma| &\geq \frac{2^{1.95k}}{(16k/h)^{3h}} \\
&= \frac{2^{1.95 \log_6 n}}{(8 \log_6 n / \log_{3/2}(n/m))^{6 \log_{3/2}(n/m)}} \\
&= \frac{n^{1.95 \log_6 2}}{(960 \log_6 n / \log_{3/2} n)^{(6 \log_{3/2} n)/120}} \\
&= \frac{n^{1.95 \log_6 2}}{(960 \log_6(3/2))^{0.05 \log_{3/2} n}} \\
&= \frac{n^{1.95 \log_6 2}}{n^{0.05 \log_{3/2}(960 \log_6(3/2))}} \\
&= n^{1.95 \log_6 2 - 0.05 \log_{3/2}(960 \log_6(3/2))} \\
&\geq n^{0.09}
\end{aligned}$$

Now suppose that an NCA labeling scheme labels the nodes of all 3-2 trees associated with sequences in Σ. Consider two trees associated with sequences $x, y \in \Sigma$, and let S denote the set of leaf labels that are common to x and y. We must then have $|S| < m$, since otherwise, by Lemma 3.2, we would have

$$\operatorname{lev}(x, y) \leq \operatorname{lev}(x, I(S)) + \operatorname{lev}(I(S), y) \leq \frac{h}{2} + \frac{h}{2} = h.$$

It follows that, if we restrict attention to a subset $\mathcal{T}$ consisting of $\min(|\Sigma|, \lfloor n/(2m) \rfloor)$ of the trees associated with strings in Σ, then the leaves of any tree in $\mathcal{T}$ can share a total of at most $n/2$ labels with all other trees in $\mathcal{T}$. In other words, every tree in $\mathcal{T}$ has at least $n/2$ leaf labels that are unique for this tree within the set of all leaf labels of trees in $\mathcal{T}$. This gives a total of at least

$$\begin{aligned} \frac{n}{2}\min(|\Sigma|, \lfloor n/(2m) \rfloor) &= \frac{n}{2}\min(n^{0.09}, \lfloor n^{1/120}/2 \rfloor) \\ &= n^{121/120}/8 \\ &\geq n^{1.008}/8 \end{aligned}$$

distinct labels. If the worst-case label size is L, we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $n^{1.008}/8 \leq 2^{L+1} - 1$ from which it follows that

$$\begin{aligned} L \geq \lfloor 1.008 \log n \rfloor - 3 &\geq \lfloor 1.008 \log(N/2 \cdot 6^{120}) \rfloor - 3 \\ &\geq 1.008 \log N - 318. \end{aligned} \qquad \square$$

4 Upper bound

In this section we construct an NCA labeling scheme that assigns to every node a label consisting of a sequence of sub-labels, each of which is constructed from a decomposition of a tree known as heavy-light decomposition. The labeling scheme is similar to that of [7] but with a different way of constructing sub-labels (presented in Section 4.4), a different way of ordering sub-labels (presented in Section 4.2) and a different way of encoding lists of sub-labels (presented in Section 4.1).

4.1 Encodings.

We begin with a collection of small results that show how to efficiently encode sequences of binary strings.

Lemma 4.1. *A collection of n objects can be uniquely labeled with binary strings of length at most L if and only if $L \geq \lfloor \log n \rfloor$.*

Proof. There are 2^L binary strings of length L, and hence there are $2^{L+1} - 1$ binary strings of length at most L. Thus, we can create unique labels for n different objects using labels of length at most L whenever $n \leq 2^{L+1} - 1$, which is equivalent to $L \geq \lceil \log(n+1) \rceil - 1 = \lfloor \log n \rfloor$. (The latter equality follows from the simple fact that $\lfloor r \rfloor = \lceil s \rceil - 1$ for all real numbers $r < s$ for which there does not exist an integer z with $r < z < s$.) □

Lemma 4.2. *A collection of n objects can be uniquely labeled with binary strings of length* exactly *L if and only if $L \geq \lceil \log n \rceil$.*

Proof. The argument is similar to the one in Lemma 4.1, but with the modification that we only use labels of length *exactly* equal to L. This yields the inequality $n \leq 2^L$, which is equivalent to $L \geq \lceil \log n \rceil$. □

Lemmas 4.1 and 4.2 can only be efficiently implemented if there is a way to efficiently implement the 1-1 correspondence between the objects and the numbers $1, \ldots, n$. The remaining lemmas of this section show how to encode sequences of binary strings whose concatenation has length t, and all of them except Lemma 4.4 can be implemented with linear time encoding and constant time decoding on a RAM machine in which a machine word has size $O(t)$.

Lemma 4.3. *Let $a = (a_1, a_2)$ be a pair of (possibly empty) binary strings with $|a_1 \cdot a_2| = t$. We can encode a as a single binary string of length $t + \lceil \log t \rceil$ such that a decoder without any knowledge of a or t can recreate a from the encoded string alone.*

Proof. Since $|a_1| \leq |a_1 \cdot a_2| = t$, we can use Lemma 4.2 to encode $|a_1|$ with exactly $\lceil \log t \rceil$ bits. We then encode a by concatenating the encoding of $|a_1|$ with $a_1 \cdot a_2$ to give a string of exactly $t + \lceil \log t \rceil$ bits. Since t is uniquely determined from $t + \lceil \log t \rceil$, the decoder can split up the encoded string into the encoding of $|a_1|$ and the concatenation $a_1 \cdot a_2$ from which it can recreate a_1 and a_2. □

We thank Mathias Bæk Tejs Knudsen for inspiring parts of the proof of Lemma 4.4 below. As the proof shows, the encoding in Lemma 4.4 is optimal with respect to size but comes with no guarantees for time complexities. Lemma 4.5 further below is a suboptimal version of Lemma 4.4 but with a more efficient implementation.

Lemma 4.4. *Let $a = (a_0, \ldots, a_{2k})$ be a list of (possibly empty) binary strings with $|a_0 \cdots a_{2k}| = t$ and with $a_{2i} \cdot a_{2i+1} \neq \varepsilon$ for all $i < k$. We can encode a as a single binary string of length $\lceil (1+\log(2+\sqrt{2}))t \rceil$ such that a decoder without any knowledge of a, t or k can recreate a from the encoded string alone.*

Proof. We will use Lemma 4.2 to encode a for a fixed t. To do this, we must count the number of possible sequences in the form of a. There are 2^t choices for the t bits in the concatenation $a_0 \cdots a_{2k}$, and every subdivision of the concatenation into the substrings a_i corresponds to a solution to the equation

$$x_0 + x_1 + \cdots + x_{2k} = t$$

where $x_{2i} + x_{2i+1} \geq 1$ for $i = 0, \ldots, k-1$. Note that we must have $k \leq t$. For a given t, let s_t denote the number of solutions (including choices of k) to the above equation. We shall prove further below that

$$s_t = \frac{1}{4}c^{t+1} + \frac{1}{4}d^{t+1}, \tag{1}$$

where $c = 2 + \sqrt{2}$ and $d = 2 - \sqrt{2}$, which easily implies $s_t \leq (2+\sqrt{2})^t$. It then follows that the total number of sequences a for fixed t is bounded by $2^t(2+\sqrt{2})^t$, and using Lemma 4.2 we can therefore encode any such a as a string with *exactly* $\lceil (1+\log(2+\sqrt{2}))t \rceil$ bits. Since t is uniquely determined by this length, the decoder can determine t from the length of the string and then use Lemma 4.2 to recreate a.

It remains to show (1). For any t, the number of solutions with $k = 0$ is 1. Given a solution where $k > 0$, let $j = x_0 + x_1$, and note that $j \geq 1$ and that $x_2 + \cdots + x_{2k} = t - j$ is a solution to the problem for $t - j$. There are $j + 1$ solutions to $x_0 + x_1 = j$, and hence the total number of solutions is

$$s_t = 1 + \sum_{j=1}^{t} s_{t-j}(j+1).$$

Using this expression, it is straightforward to see that

$$s_t - 2s_{t-1} + s_{t-2} = 2s_{t-1} - s_{t-2},$$

which implies $s_t = 4s_{t-1} - 2s_{t-2}$. The characteristic polynomial of this recurrence relation has roots c and d, and hence $s_t = \alpha c^t + \beta d^t$ for some α, β. Using $s_0 = 1$ and $s_1 = 3$ to solve, we obtain $\alpha = c/4$ and $\beta = d/4$, which proves (1). □

Lemma 4.5. *Let $a = (a_0, \ldots, a_{2k})$ be a list of (possibly empty) binary strings with $|a_0 \cdots a_{2k}| = t$ and with $a_{2i} \cdot a_{2i+1} \neq \varepsilon$ for all $i < k$. We can encode a as a single binary string of length $3t$ such that a decoder without any knowledge of a, t or k can recreate a from the encoded string alone.*

Proof. We encode a as a concatenation of three binary strings of lengths t, $t-1$ and $t+1$, respectively. The first string is the concatenation $\tilde{a} = a_0 \cdots a_{2k}$. The second string has a $\texttt{1}$ in the i'th position for $i \leq t-1$ exactly when the $(i+1)$'th position of $\tilde{a}$ is the first bit in a substring $a_{2j} \cdot a_{2j+1}$ (which by the assumption is nonempty for all j). The third string has a $\texttt{1}$ in the i'th position for $i \leq t$ exactly when the i'th position of $\tilde{a}$ is the first bit in a substring a_{2j+1} for some j or in a_{2k}, and a $\texttt{1}$ in the $(t+1)$'th position exactly when $a_{2k} \neq \varepsilon$.

If the decoder receives the concatenation of length $3t$ of these three strings, it can easily recreate the three strings by splitting up the string into three substrings of sizes t, $t-1$ and $t+1$. The first string is $\tilde{a}$, which it can then split up at all positions where the second string has a $\texttt{1}$. This gives a list of nonempty strings in the form $a_{2i} \cdot a_{2i+1}$ for $i \leq k-2$ as well as the string $a_{2k-2} \cdot a_{2k-1} \cdot a_{2k}$. The decoder can then use the third string to split up each of these concatenations as follows. For every (nonempty) concatenation $a_{2i} \cdot a_{2i+1}$, consider the corresponding bits in the third string. If one of these bits is a $\texttt{1}$, then the concatenation should be split up at that position; in particular, if the $\texttt{1}$ is at the first bit in the concatenation, then it means that a_{2i} is empty. If none of the bits is a $\texttt{1}$, then it means that a_{2i+1} is empty. In all cases, we can recreate a_{2i} and a_{2i+1}. Likewise, the concatenation $a_{2k-2} \cdot a_{2k-1} \cdot a_{2k}$ can be split up using the $\texttt{1}$s in the corresponding bits in the third string. If there are two $\texttt{1}$s among these bits, then it is clear how to split up the concatenation. If there are no $\texttt{1}$s, then it means that a_{2k-1} and a_{2k} are both empty. If there is exactly one $\texttt{1}$, then we can split up the concatenation into a_{2k-2} and $a_{2k-1} \cdot a_{2k}$, and exactly one of a_{2k-1} and a_{2k} must be empty. The last bit of the third string determines which of these two cases we are in. □

Lemma 4.6. *Let $a = (a_0, \ldots, a_k)$ be a list of (possibly empty) binary strings with $|a_0 \cdots a_k| = t$ and with $a_i \cdot a_{i+1} \neq \varepsilon$ for all $i < k$. We can encode a as a single binary string of length $\lceil (1 + \log 3)(t-1) \rceil + 3$ such that a decoder without any knowledge of a, t or k can recreate a from the encoded string alone.*

Proof. We encode a by concatenating $\tilde{a} = a_0 \cdots a_k$ of length t with a string s of length $\lceil (\log 3) t \rceil$. To describe s, we first construct a string $\tilde{s}$ of length $t-1$ over the alphabet $\{\texttt{0}, \texttt{1}, \texttt{2}\}$. The i'th bit $\tilde{s}_i$ of $\tilde{s}$ is defined according to the role of the $(i+1)$'th bit x in $\tilde{a}$ as follows:

$$\tilde{s}_i = \begin{cases} \texttt{0}, & \text{if } x \text{ is the first bit of a nonempty string } a_j, \\ & \text{where } a_{j-1} \text{ is nonempty,} \\ \texttt{1}, & \text{if } x \text{ is the first bit of a nonempty string } a_j, \\ & \text{where } a_{j-1} \text{ is empty,} \\ \texttt{2}, & \text{else.} \end{cases}$$

The string $\tilde{s}$ represents a unique choice out of 3^{t-1} possibilities, and by Lemma 4.2 we can represent this choice with a binary string s of length exactly equal to $\lceil \log 3^{t-1} \rceil = \lceil (t-1) \log 3 \rceil$. We concatenate this with a single indicator bit representing whether a_0 is empty or not, and another indicator bit representing whether a_k is empty or not. Finally, we concatenate all this with $\tilde{a}$, giving a string of total length $\lceil (t-1) \log 3 \rceil + 2 + t = \lceil (1 + \log 3)(t-1) \rceil + 3$.

Since the value t is uniquely determined from the length of the encoded string, the decoder is able to split up the encoded string into $\tilde{a}$, s and the two indicator bits. It can then convert s to $\tilde{s}$ and use the entries in $\tilde{s}$ and the indicator bits to recreate a from $\tilde{a}$. This proves the theorem. □

4.2 An order on binary strings.

Consider the total order $\preceq$ on binary strings defined by

$$s \cdot \mathtt{0} \cdot t \prec s \prec s \cdot \mathtt{1} \cdot t'$$

for all binary strings s, t, t'. Here we have written $s \prec t$ as short for $s \preceq t \wedge s \neq t$. This order naturally arises in many contexts and has been studied before; see, for example, [57]. All binary strings of length three or less are ordered by $\preceq$ as follows:

$$\mathtt{000} \prec \mathtt{00} \prec \mathtt{001} \prec \mathtt{0} \prec \mathtt{010} \prec \mathtt{01} \prec \mathtt{011} \prec \varepsilon \prec \mathtt{100} \prec \mathtt{10} \prec \mathtt{101} \prec \mathtt{1} \prec \mathtt{110} \prec \mathtt{11} \prec \mathtt{111}$$

A finite sequence (a_i) of binary strings is *$\prec$-ordered* if $a_i \prec a_j$ for $i < j$.

Lemma 4.7. *Given a finite sequence (w_i) of positive numbers with $w = \sum_i w_i$, there exists an $\prec$-ordered sequence (a_i) with $|a_i| \leq \lfloor \log w - \log w_i \rfloor$ for all i.*

Proof. The proof is by induction on the number of elements in the sequence (w_i). If there is only one element, w_1, then we can set $a_1 = \varepsilon$, which satisfies $|a_1| = 0 = \lfloor \log w_1 - \log w_1 \rfloor$. So suppose that there is more than one element in the sequence and that the theorem holds for shorter sequences. Let k be the smallest index such that $\sum_{i \leq k} w_i > w/2$, and set $a_k = \varepsilon$. Then a_k clearly satisfies the condition. The subsequences $(w_i)_{i<k}$ and $(w_i)_{i>k}$ are shorter and satisfy $\sum_{i<k} w_i \leq w/2$ and $\sum_{i>k} w_i \leq w/2$, so by induction there exist $\prec$-ordered sequences $(b_i)_{i<k}$ and $(b_i)_{i>k}$ with $|b_i| \leq \lfloor \log(w/2) - \log w_i \rfloor = \lfloor \log w - \log w_i \rfloor - 1$ for all $i \neq k$. Now, define a_i for $i < k$ by $a_i = \mathtt{0} \cdot b_i$ and for $i > k$ by $a_i = \mathtt{1} \cdot b_i$. Then (a_i) is a $\prec$-ordered sequence with $|a_i| \leq \lfloor \log w - \log w_i \rfloor$ for all i. □

A linear time implementation of the previous lemma can be achieved as follows. First compute the numbers $t_i = \lfloor \log w - \log w_i \rfloor$ in linear time. Now set $a_1 = \mathtt{0}^{t_1}$ to be the minimum (with respect to the order $\preceq$) binary string of length at most t_1. At the i'th step, set a_i to be the minimum binary string of length at most t_i with $a_{i-1} \prec a_i$. If this process successfully terminates, then the sequence (a_i) has the desired property. On the other hand, the process must terminate, because the above lemma says that there *exists* an assignment of the a_i's, and our algorithm conservatively chooses each a_i so that the set of possible choices left for a_{i+1} is maximal at every step. A similar argument shows that the following lemma can be implemented in linear time.

Lemma 4.8. *Given a finite sequence (w_i) of positive numbers with $w = \sum_i w_i$, there exist an $\prec$-ordered sequence (a_i) of nonempty strings and a k such that $|a_i| \leq \lfloor \log(w + w_k) - \log w_i \rfloor$ for all i.*

Proof. Let k be the smallest index such that $\sum_{i \leq k} w_i > w/2$ and add an extra copy of w_k next to w_k in the sequence of weights. The total sequence of weights will now sum to $w + w_k$, and if we apply Lemma 4.7 to this sequence, exactly one of the two copies of w_k will be assigned the empty string. Discard this string, and what is left is a $\prec$-ordered sequence (a_i) with $|a_i| \leq \lfloor \log(w + w_k) - \log w_i \rfloor$ for all i as desired. □

4.3 Heavy-light decomposition.

We next describe the *heavy-light decomposition* of Harel and Tarjan [38]. Let T be a rooted tree. The nodes of T are classified as either *heavy* or *light* as follows. The root r of T is light. For each internal node v, pick one child node w where $\mathrm{size}(w)$ is maximal among the children of v and classify it as heavy; classify the other children of v as light. We denote the unique heavy child of v by $\mathrm{hchild}(v)$ and the set of light children by $\mathrm{lchildren}(v)$. The *light size* of a node v is the number $\mathrm{lsize}(v) = 1+\sum_{w\in \mathrm{lchildren}(v)} \mathrm{size}(w)$, which is equal to $\mathrm{size}(v)-\mathrm{size}(\mathrm{hchild}(v))$ when v is internal. The *apex* of v, denoted $\mathrm{apex}(v)$, is the nearest light ancestor of v. By removing the edges between light nodes and their parents, T is divided into a collection of *heavy paths*. The set of nodes on the same heavy path as v is denoted $\mathrm{hpath}(v)$. The top node of $\mathrm{hpath}(v)$ is the light node $\mathrm{apex}(v)$.

For a node v, consider the sequence $u_0, \ldots, u_k$ of light nodes encountered on the path from the root $r = u_0$ to v. The number k is the *light depth* of v, denoted $\mathrm{ldepth}(v)$. The light depth of T, $\mathrm{ldepth}(T)$ is the maximum light depth among the nodes in T. Note that $\mathrm{ldepth}(v) \leq \mathrm{ldepth}(T) \leq \log n$; see [38].

4.4 One NCA labeling scheme.

We now describe the labeling scheme that will be used for various families of trees, although with different encodings for each family. Given a rooted tree T, we begin by assigning to each node v a *heavy label*, $\mathrm{hlabel}(v)$, and, when v is light and not equal to the root, a *light label*, $\mathrm{llabel}(v)$, as described in Lemmas 4.9 and 4.10 below.

Lemma 4.9. *There exist binary strings* $\mathrm{hlabel}(v)$ *for all nodes* v *in* T *so that the following hold for all nodes* v, w *belonging to the same heavy path:*

$$\begin{gathered}\mathrm{depth}(v) < \mathrm{depth}(w) \implies \\ \mathrm{hlabel}(v) \prec \mathrm{hlabel}(w)\end{gathered} \tag{2}$$

$$|\,\mathrm{hlabel}(v)| \leq \lfloor \log \mathrm{size}(\mathrm{apex}(v)) - \log \mathrm{lsize}(v)\rfloor \tag{3}$$

Proof. Consider each heavy path H separately and use the sequence $(\mathrm{lsize}(v))_{v\in H}$, ordered ascendingly by $\mathrm{depth}(v)$, as input to Lemma 4.7. □

Lemma 4.10. *There exist binary strings* $\mathrm{llabel}(v)$ *for all light nodes* $v \neq r$ *in* T *so that the following hold for all light siblings* v, w*:*

$$v \neq w \implies \mathrm{llabel}(v) \neq \mathrm{llabel}(w) \tag{4}$$

$$|\,\mathrm{llabel}(v)| \leq \lfloor \log \mathrm{lsize}(\mathrm{parent}(v)) - \log \mathrm{size}(v)\rfloor \tag{5}$$

Proof. Consider each set L of light siblings separately and use the sequence $(\mathrm{size}(v))_{v\in L}$, not caring about order, as input to Lemma 4.7. □

In many cases we are not going to use the constructions in Lemmas 4.9 and 4.10 directly, but will instead use the following two modifications:

Lemma 4.11. *It is possible to modify the constructions in Lemmas 4.9 and 4.10 so that, for all nodes* u, v *where* v *is a light child of* u*,*

$$\mathrm{hlabel}(u) = \varepsilon \implies \mathrm{llabel}(v) \neq \varepsilon. \tag{6}$$

The modification still satisfies (2)*,* (3)*,* (4) *and* (5) *except that when* $\mathrm{hlabel}(u)$ *is empty,* (5) *is replaced by*

$$|\,\mathrm{hlabel}(u)| + |\,\mathrm{llabel}(v)| \leq \lfloor \log \mathrm{size}(\mathrm{apex}(u)) - \log \mathrm{size}(v)\rfloor \tag{7}$$

Proof. First observe that without modifying the construction in Lemmas 4.9 and 4.10 we can combine (3) with (5) to obtain (7). We now describe the modification: the construction works exactly as in the two lemmas except that in cases where $\text{hlabel}(u)$ is empty, we use Lemma 4.8 in place of Lemma 4.7 in the construction of light labels in Lemma 4.10. This clearly makes (6) true, so it remains to prove (7).

So let u and v be as above. By construction of the heavy-light decomposition, $\text{size}(\text{hchild}(u))$ is larger than or equal to the size of any of the light children of u, and hence larger than the size that corresponds to the weight w_k in Lemma 4.8. Further, $\text{lsize}(u) + \text{size}(\text{hchild}(u)) = \text{size}(u) \leq \text{size}(\text{apex}(u))$. Using these two facts together, Lemma 4.8 now yields

$$|\,\text{llabel}(v)| \leq \lfloor \log \text{size}(\text{apex}(u)) - \log \text{size}(v) \rfloor.$$

Since $|\,\text{hlabel}(u)| = 0$, we have therefore obtained (7). □

Lemma 4.12. *It is possible to modify the constructions in Lemmas 4.9 and 4.10 so that, for all nodes u, v, w where v is a light child of u and w is a descendant of v on the same heavy path as v,*

$$\text{hlabel}(u) = \varepsilon \text{ and } \text{llabel}(v) = \varepsilon \implies \text{hlabel}(w) \neq \varepsilon. \tag{8}$$

The modification still satisfies (2), (3), (4) *and* (5) *except that when* $\text{hlabel}(u)$ *and* $\text{llabel}(v)$ *are both empty,* (3) *is replaced by*

$$|\,\text{hlabel}(u)| + |\,\text{llabel}(v)| + |\,\text{hlabel}(w)| \leq \lfloor \log \text{size}(\text{apex}(u)) - \log \text{lsize}(w) \rfloor \tag{9}$$

Proof. The proof is similar to that of the previous lemma. First observe that without modifying the construction in Lemmas 4.9 and 4.10 we can combine (3), (5) and (3) again to obtain (9). We now describe the modification: the construction works exactly as in the two lemmas except that in cases where $\text{hlabel}(u)$ and $\text{llabel}(v)$ are both empty, we use Lemma 4.8 in place of Lemma 4.7 in the construction of heavy labels in Lemma 4.9. This clearly makes (8) true, so it remains to prove (9).

So let u, v and w be as above. Note that $\text{size}(v)$ is larger than or equal to the light size of any of the nodes on the heavy path with v as apex, and hence larger than the light size that corresponds to the weight w_k in Lemma 4.8. Further, $2\,\text{size}(v) \leq \text{lsize}(u) + \text{size}(\text{hchild}(u)) = \text{size}(u)) \leq \text{size}(\text{apex}(u))$. Using these two facts together, Lemma 4.8 now yields

$$|\,\text{hlabel}(w)| \leq \lfloor \log \text{size}(\text{apex}(u)) - \log \text{lsize}(w) \rfloor.$$

Since $|\,\text{hlabel}(u)| = |\,\text{llabel}(v)| = 0$, we have therefore obtained (9). □

We next assign a new set of labels for the nodes of T. Given a node v with $\text{ldepth}(v) = k$, consider the sequence $u_0, v_0, \ldots, u_k, v_k$ of nodes from the root $r = u_0$ to $v = v_k$, where $u_i = \text{apex}(v_i)$ is light for $i = 0, \ldots, k$ and $v_{i-1} = \text{parent}(u_i)$ for $i = 1, \ldots, k$. Let $l(v) = (h_0, l_1, h_1, \ldots, l_k, h_k)$, where $l_i = \text{llabel}(u_i)$ and $h_i = \text{hlabel}(v_i)$. Figure 1 shows an example of a tree with the labels $l(v)$. Note that we have used Lemmas 4.9 and 4.10 for the construction of labels in this figure and not any of the modifications in Lemmas 4.11 and 4.12.

To define a labeling scheme, it remains to encode the lists $l(v)$ of binary strings into a single binary string. Before we do this, however, we note that $l(\text{nca}(v, w))$ can be computed directly from $l(v)$ and $l(w)$. The proof is essentially the same as that in [7] although with the order $\preceq$ in place of the usual lexicographic order.

Lemma 4.13. *Let v and w be nodes in T, and let $u = \text{nca}(v, w)$.*

(a) If $l(v)$ is a prefix of $l(w)$, then $l(u) = l(v)$.

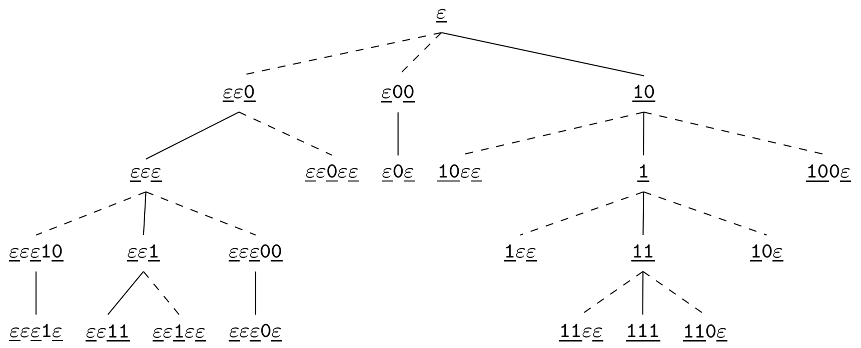

Figure 1: A tree with the labels $l(v)$ from Section 4.4 and with heavy sub-labels underlined.

(b) If $l(w)$ is a prefix of $l(v)$, then $l(u) = l(w)$.

(c) If $l(v) = (h_0, l_1, \dots, h_i, l_i, \dots)$ and $l(w) = (h_0, l_1, \dots, h_i, l'_i, \dots)$ with $l_i \neq l'_i$, then $l(u) = (h_0, l_1, h_1, \dots, h_i)$.

(d) If $l(v) = (h_0, l_1, \dots, l_{i-1}, h_i, \dots)$ and $l(w) = (h_0, l_1, h_1, \dots, l_{i-1}, h'_i, \dots)$ with $h_i \neq h'_i$, then $l(u) = (h_0, l_1, h_1, \dots l_{i-1}, \min_{\preceq}\{h_i, h'_i\})$.

Proof. By construction, $l = l(\text{parent}(\text{apex}(u)))$ is a prefix of both $l(v)$ and $l(w)$, and

$$l(u) = l \cdot (\text{llabel}(\text{apex}(u)), \text{hlabel}(u)).$$

Suppose first that v is an ancestor of w, so that $u = v$, and let x be the nearest ancestor of w on $\text{hpath}(v)$. Then $\text{apex}(x) = \text{apex}(u)$, so

$$l(w) = l \cdot (\text{llabel}(\text{apex}(u)), \text{hlabel}(x), \dots)$$

If $u = x$, then $\text{hlabel}(x) = \text{hlabel}(u)$ and case (a) applies. (If $v = w$ then case (b) applies too.) If $u \neq x$, then $\text{hlabel}(u) \prec \text{hlabel}(x)$ by (2) and case (d) applies. The case where w is an ancestor of v is analogous.

Suppose next that v and w are not ancestors of each other. Then u must have children $\hat{v}$ and $\hat{w}$ with $\hat{v} \neq \hat{w}$ such that $\hat{v}$ is an ancestor of v and $\hat{w}$ is an ancestor of w. At most one of $\hat{v}$ and $\hat{w}$ can be heavy. If neither of them are heavy, then they are apexes for their own heavy paths, and hence

$$l(v) = l(u) \cdot (\text{llabel}(\hat{v}), \dots)$$

and

$$l(w) = l(u) \cdot (\text{llabel}(\hat{w}), \dots).$$

By (4), $\text{llabel}(\hat{v})$ and $\text{llabel}(\hat{w})$ are distinct, so case (c) applies. If $\hat{v}$ is heavy, then $\text{apex}(\hat{v}) = \text{apex}(u)$ and $l(v) = l \cdot (\text{llabel}(\text{apex}(u)), \text{hlabel}(\hat{v}), \dots)$ while $l(w)$ is still on the above form, i.e. $l(w) = l \cdot (\text{llabel}(\text{apex}(u)), \text{hlabel}(u), \dots)$. By (2), $\text{hlabel}(u) \prec \text{hlabel}(\hat{v})$, so (d) applies. The case where $\hat{w}$ is heavy is analogous. □

Note that, as in [7], the above theorem can be used to find labels for NCAs in constant time on the RAM as long as the labels have size $O(\log n)$.

As a final step, before presenting the encodings of the labels $l(v)$, we present a lemma that makes it easier to compute the size of the encodings. For brevity, we let $\tilde{l}(v) = h_0 \cdot l_1 \cdot h_1 \cdots l_k \cdot h_k$ denote the concatenation of the sub-labels of $l(v)$.

Lemma 4.14. *If T has n nodes, then $|\tilde{l}(v)| \leq \lfloor \log n \rfloor$ for every node v in T. This holds no matter if we use Lemmas 4.9 and 4.10 combined or any of the variants in Lemmas 4.11 and 4.12 for the construction of heavy and light labels.*

Proof. Let v be an arbitrary node in T and recall that $l(v) = (h_0, l_1, h_1, \dots, l_k, h_k)$ where $l_i = \text{llabel}(u_i)$ and $h_i = \text{hlabel}(v_i)$ for nodes u_i, v_i, $i = 0, \dots, k$ given by $r = u_0$, $v = v_k$, $u_i = \text{apex}(v_i)$ for all $i = 0, \dots, k$ and $v_{i-1} = \text{parent}(u_i)$ for $i = 1, \dots, k$. If we use Lemmas 4.9 and 4.10 for the construction of heavy and light labels, we have by (3) that $|h_i| \leq \lfloor \log \text{size}(u_i) - \log \text{lsize}(v_i) \rfloor$ for all $i = 0, \dots, k$ and by (5) that $|l_i| \leq \lfloor \log \text{lsize}(v_{i-1}) - \log \text{size}\, u_i \rfloor$ for $i = 1, \dots, k$. Summarizing now gives a telescoping sum:

$$\begin{aligned}
|\tilde{l}(v)| &= |h_0 \cdot l_1 \cdot h_1 \cdots l_k \cdot h_k| \\
&\leq \lfloor \log \text{size}(u_0) - \log \text{lsize}(v_0) \rfloor + \\
&\qquad \lfloor \log \text{lsize}(v_0) - \log \text{size}(u_1) \rfloor + \\
&\qquad \cdots + \lfloor \log \text{size}(u_k) - \log \text{lsize}(v_k) \rfloor \\
&\leq \lfloor \log \text{size}(u_0) - \log \text{lsize}(v_k) \rfloor \\
&\leq \lfloor \log n \rfloor.
\end{aligned}$$

In the cases where we have used any of the variants in Lemmas 4.11 and 4.12, we must use (7) or (9) first to collapse sums of two or three terms in the above sum before collapsing the whole expression. Nevertheless, the result of the computation remains unchanged. □

4.5 NCA labeling schemes for different families of trees.

Let Trees and BinaryTrees denote the families of rooted trees and binary trees, respectively.

Theorem 4.15. *There exists an NCA labeling scheme for* Trees *whose worst-case label size is at most $\lceil (1 + \log(2 + \sqrt{2})) \lfloor \log n \rfloor \rceil \leq 2.772 \log n + 1$.*

Proof. The encoder uses the modified construction in Lemma 4.11 to ensure that every empty heavy label is followed by a nonempty light label. This means that the sequence $l(v) = (h_0, l_1, h_1, \dots, l_k, h_k)$ can be encoded using $\lceil (1 + \log(2 + \sqrt{2})) \lfloor \log n \rfloor \rceil$ bits; see Lemma 4.4. Given the encoded labels from two nodes, the decoder can now decode the labels as described in Lemma 4.4, use Lemma 4.13 to compute the label of the NCA, and then re-encode that label using Lemma 4.4 once again. □

The labeling scheme in Theorem 4.15 makes use of Lemma 4.4 which comes without any guarantees for the time complexities for encoding and decoding. This makes the result less applicable in practice. Theorems 4.16, 4.18 and 4.19 and Corollary 4.17 below all use linear time for encoding and constant time for decoding.

Theorem 4.16. *There exists an NCA labeling scheme for* Trees *whose worst-case label size is at most $3 \lfloor \log n \rfloor$.*

Proof. The proof is identical to that of Theorem 4.15 but with Lemma 4.5 in place of Lemma 4.4. □

A variant of NCA labeling schemes [13] allows every node to also have a *predefined* label and requires the labeling scheme to return the predefined label of the NCA.

Corollary 4.17. *There exists an NCA labeling scheme for* Trees *with predefined labels of fixed length k whose worst-case label size is at most $(3 + k) \lfloor \log n \rfloor + 1$.*

Proof. It suffices to save together with the NCA label of a node v a table of the predefined labels for the at most $\lfloor \log n \rfloor$ parents of light nodes on the path from the root to v, since the NCA of two nodes will always be a such for one of the nodes. By prepending a string in the form $\mathtt{0}^i\mathtt{1}$ to the NCA label of v we can ensure that it has size *exacly* $3\lfloor \log n \rfloor + 1$. We can then append a table of up to $\lfloor \log n \rfloor$ predefined labels of size k. Finally, we append $\mathtt{0}$s to make the label have size exactly $(3 + k)\lfloor \log n \rfloor + 1$. The decoder can now use the label's length to split up the label into the NCA label and the entries in the table of predefined labels. □

Theorem 4.18. *There exists an NCA labeling scheme for* BinaryTrees *whose worst-case label size is at most* $\lceil (1 + \log 3)(\lfloor \log n \rfloor - 1) \rceil + 3 \leq 2.585 \log n + 2$.

Proof. First note that every node in a binary tree has at most one light child. We can therefore assume that all light labels are empty. Letting the encoder use the construction in Lemma 4.12, we can then ensure that every empty heavy label is followed by (an empty light label and) a nonempty heavy label. Since we can ignore light labels, it suffices to encode the sequence $(h_0, h_1, \ldots, h_k)$, and this sequence can be encoded with $\lceil (1 + \log 3)(\lfloor \log n \rfloor - 1) \rceil + 3$ bits; see Lemma 4.6. The rest of the proof follows the same argument as the proof of Theorem 4.15. □

A *caterpillar* is a tree in which all leaves are connected to a single *main path*. We assume caterpillars to always be rooted at one of the end nodes of the main path. Let Caterpillars denote the family of caterpillars.

Theorem 4.19. *There exists an NCA labeling scheme for* Caterpillars *whose worst-case label size is at most* $\lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil + 1$.

Proof. By definition of caterpillars, every label $l(v)$ is either in the form (h_0) or (h_0, l_1, ε). We encode the first case as $\mathtt{0} \cdot h_0$ and the second case as $\mathtt{1} \cdot x$, where x is the encoding of the pair (h_0, l_1) using $\lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil$ bits; see Lemma 4.3. In both cases, the label size is at most $\lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil + 1$, and the decoder can easily distinguish the two cases from the first bit. The rest of the proof follows the same argument as the proof of Theorem 4.15. □

For comparison, the best known lower bound for NCA labeling schemes for caterpillars is the trivial $\lfloor \log n \rfloor$.

References

[1] S. Abiteboul, S. Alstrup, H. Kaplan, T. Milo, and T. Rauhe, *Compact labeling scheme for ancestor queries*, SIAM J. Comput. **35** (2006), no. 6, 1295–1309.

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

[3] A. V. Aho, Y. Sagiv, T. G. Szymanski, and J. D. Ullman, *Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions*, SIAM Journal on Computing **10** (1981), no. 3, 405–421.

[4] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[5] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *On finding lowest common ancestor in trees*, SIAM Journal on computing **5** (1976), no. 1, 115–132, See also STOC 1973.

[6] S. Alstrup, P. Bille, and T. Rauhe, *Labeling schemes for small distances in trees*, SIAM J. Discrete Math. **19** (2005), no. 2, 448–462.

[7] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, *Nearest common ancestors: A survey and a new algorithm for a distributed environment*, Theory of Computing Systems **37** (2004), no. 3, 441–456.

[8] S. Alstrup and T. Rauhe, *Improved labeling schemes for ancestor queries*, Proc. of the 13th annual ACM-SIAM Symp. on Discrete Algorithms (SODA), 2002.

[9] ______, *Small induced-universal graphs and compact implicit graph representations*, In Proc. 43rd annual IEEE Symp. on Foundations of Computer Science, 2002, pp. 53–62.

[10] S. Alstrup and M. Thorup, *Optimal pointer algorithms for finding nearest common ancestors in dynamic trees*, Journal of Algorithms **35** (2000), no. 2, 169–188.

[11] M. A. Bender and M. Farach-Colton, *The lca problem revisted*, 4th LATIN, 2000, pp. 88–94.

[12] O. Berkman and U. Vishkin, *Recursive star-tree parallel data structure*, SIAM Journal on Computing **22** (1993), no. 2, 221–242.

[13] L. Blin, S. Dolev, M. Potop-Butucaru, and S. Rovedakis, *Fast self-stabilizing minimum spanning tree construction: using compact nearest common ancestor labeling scheme*, Proceedings of the 24th international conference on Distributed computing, DISC'10, 2010, pp. 480–494.

[14] N. Bonichon, C. Gavoille, and A. Labourel, *Short labels by traversal and jumping*, Electronic Notes in Discrete Mathematics **28** (2007), 153–160.

[15] M. A. Breuer, *Coding vertexes of a graph*, IEEE Trans. on Information Theory **IT–12** (1966), 148–153.

[16] M. A. Breuer and J. Folkman, *An unexpected result on coding vertices of a graph*, J. of Mathemathical analysis and applications **20** (1967), 583–600.

[17] S. Caminiti, I. Finocchi, and R. Petreschi, *Engineering tree labeling schemes: A case study on least common ancestors.*, ESA, Lecture Notes in Computer Science, vol. 5193, Springer, 2008, pp. 234–245.

[18] S. Carlsson and B. J. Nilsson, *Computing vision points in polygons*, Algorithmica **24** (1999), no. 1, 50–75.

[19] S. Chaudhuri and C. D. Zaroliagis, *Shortest paths in digraphs of small treewdith. Part II: Optimal parallel algorithms*, Theoretical Computer Science **203** (1998), no. 2, 205–223.

[20] E. Cohen, H. Kaplan, and T. Milo, *Labeling dynamic xml trees*, SIAM J. Comput. **39** (2010), no. 5, 2048–2074.

[21] R. Cole and R. Hariharan, *Dynamic lca queries on trees*, Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), vol. 10, 1999.

[22] L. J. Cowen, *Compact routing with minimum stretch*, Journal of Algorithms **38** (2001), 170–183.

[23] B. Dixon, M. Rauch, and R. E. Tarjan, *Verification and sensitivity analysis of minimum spanning trees in linear time*, SIAM Journal on Computing **21** (1992), no. 6, 1184–1192.

[24] T. Eilam, C. Gavoille, and D. Peleg, *Compact routing schemes with low stretch factor*, 17^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), August 1998, pp. 11–20.

[25] M. Farach-Colton, *Optimal suffix tree construction with large alphabets*, 38th Annual Symposium on Foundations of Computer Science (IEEE, ed.), IEEE Computer Society Press, 1997, pp. 137–143.

[26] M. Farach-Colton, S. Kannan, and T. Warnow, *A robust model for finding optimal evolutionary trees.*, Algorithmica **13** (1995), no. 1/2, 155–179.

[27] J. Fischer, *Short labels for lowest common ancestors in trees*, ESA, 2009, pp. 752–763.

[28] P. Flocchini, T. Mesa Enriquez, L. Pagli, G. Prencipe, and N. Santoro, *Distributed minimum spanning tree maintenance for transient node failures*, IEEE Trans. Comput. **61** (2012), no. 3, 408–414.

[29] P. Fraigniaud and C. Gavoille, *Routing in trees*, 28^{th} International Colloquium on Automata, Languages and Programming (ICALP), vol. 2076 of LNCS, 2001, pp. 757–772.

[30] P. Fraigniaud and C. Gavoille., *A space lower bound for routing in trees*, 19^{th} Annual Symposium on Theoretical Aspects of Computer Science (STACS), March 2002, pp. 65–75.

[31] P. Fraigniaud and A. Korman, *Compact ancestry labeling schemes for xml trees*, SODA, 2010, pp. 458–466.

[32] ______, *An optimal ancestry scheme and small universal posets*, Proceedings of the 42nd ACM symposium on Theory of computing (New York, NY, USA), 2010, pp. 611–620.

[33] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, *Scaling and related techniques for geometry problems*, Proc. of the Sixteenth Annual ACM Symposium on Theory of Computing, 1984, pp. 135–143.

[34] H.N. Gabow, *Data structure for weighted matching and nearest common ancestors with linking*, Annual ACM-SIAM Symposium on discrete algorithms (SODA), vol. 1, 1990, pp. 434–443.

[35] C. Gavoille and D. Peleg, *Compact and localized distributed data structures*, Distributed Computing **16** (2003), no. 2-3, 111–120.

[36] C. Gavoille, D. Peleg, S. Perennes, and R. Raz, *Distance labeling in graphs*, 12th Symp. On Discrete algorithms, 2001.

[37] D. Gusfield, *Algorithms on strings, trees, and sequences*, Cambridge University Press, 1997, pp. 196-207.

[38] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, Siam J. Comput **13** (1984), no. 2, 338–355.

[39] T. C. Hu and A. C. Tucker, *Optimum computer search trees*, SIAM Journal of Applied Mathematics **21** (1971), 514–532.

[40] S. Kannan, M. Naor, and S. Rudich, *Implicit representation of graphs*, SIAM J. DISC. MATH. (1992), 596–603, Preliminary version appeared in STOC'88.

[41] H. Kaplan and T. Milo, *Short and distances and other functions*, 7nd Work. on Algo. and Data Struc., LNCS, 2001.

[42] H. Kaplan, T. Milo, and R. Shabo, *A comparison of labeling schemes for ancestor queries*, Proceedings of the thirteen annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002.

[43] D. R. Karger, P. N. Klein, and R. E. Tarjan, *A randomized linear-time algorithm to find minimum spanning trees*, Journal of the ACM **42** (1995), no. 2, 321–328.

[44] M. Katz, N. Katz, and D. Peleg, *Distance labeling schemes for well-seperated graph classes*, STACS'00, LNCS, vol. 1170, Springer Verlag, 2000.

[45] M. Katz, N. A. Katz, A. Korman, and D. Peleg, *Labeling schemes for flow and connectivity*, SIAM J. Comput. **34** (2004), no. 1, 23–40.

[46] A. Korman and S. Kutten, *Labeling schemes with queries.*, SIROCCO, 2007, pp. 109–123.

[47] V. I. Levenshtein, *Binary codes capable of correcting deletions, insertions and reversals.*, Soviet Physics Doklady. **10** (1966), no. 8, 707–710.

[48] D. Maier, *A space efficient method for the lowest common ancestor problem and an application to finding negative cycles*, 18th Annual Symposium on Foundations of Computer Science, 1977, pp. 132–141.

[49] K. Mehlhorn, *A best possible bound for the weighted path length of binary search trees*, SIAM J. Comput. **6** (1977), no. 2, 235–239.

[50] L. Pagli, G. Prencipe, and T. Zuva, *Distributed computation for swapping a failing edge*, Proceedings of the 6th international conference on Distributed Computing (Berlin, Heidelberg), IWDC'04, Springer-Verlag, 2004, pp. 28–39.

[51] D. Peleg, *Proximity-preserving labeling schemes and their applications*, Graph-Theoretic concepts in computer science, 25th international workshop WG'99, LNCS, vol. 1665, Springer Verlag, 1999, pp. 30–41.

[52] ———, *Informative labeling schemes for graphs*, 25^{th} International Symposium on Mathematical Foundations of Computer Science (MFCS), vol. 1893 of LNCS, Springer, August 2000, pp. 579–588.

[53] P. Powel, *A further improved lca algorithm*, Tech. Report TR90-01, University of Minneapolis, 1990.

[54] H. Robbins, *A remark on Stirling's formula*, Amer. Math. Monthly **62** (1955), 26–29. MR MR0069328 (16,1020e)

[55] N. Santoro and R. Khatib, *Labeling and implicit routing in networks*, The computer J. **28** (1985), 5–8.

[56] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM Journal of Computing **17** (1988), 1253–1262.

[57] M. Thorup and U. Zwick, *Compact routing schemes*, ACM Symposium on Parallel Algorithms and Architectures, vol. 13, 2001.

[58] A. K. Tsakalidis, *Maintaining order in a generalized linked list*, Acta Informatica **21** (1984), no. 1, 101–112.

[59] J. Westbrook, *Fast incremental planarity testing*, Automata, Languages and Programming, 19th International Colloquium (Werner Kuich, ed.), Lecture Notes in Computer Science, vol. 623, Springer-Verlag, 1992, pp. 342–353.