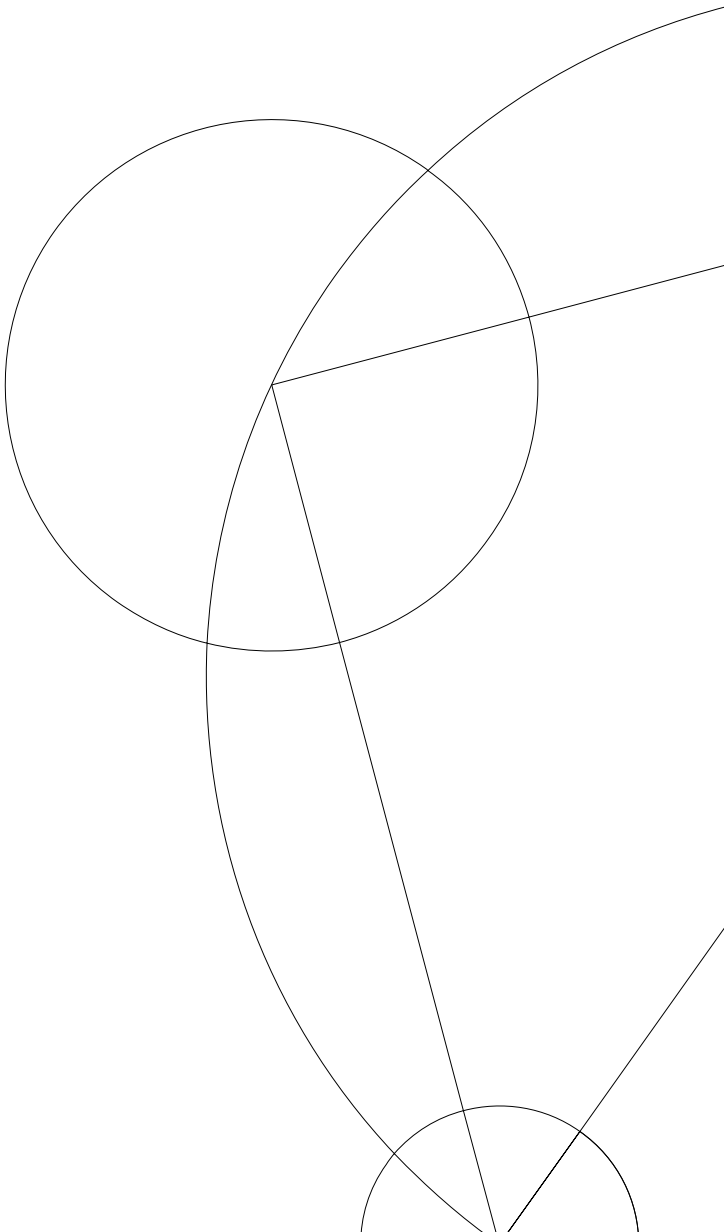
FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master thesis

Esben Bistrup Halvorsen

Labeling schemes for trees

Overview and new results



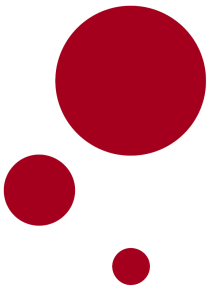
Advisors: Stephen Alstrup and Christian Wulff-Nilsen

Submitted: July 19, 2013

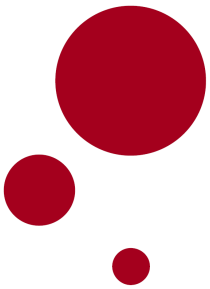
Abstract

A labeling scheme is an algorithm that labels the nodes of a graph so that information regarding a small subset of the graph can be inferred directly from the labels of the nodes in the subset. The two main characteristics of a labeling scheme are: the type of queries that can be answered from the labels; and the family of graphs which the labeling scheme can accept as input. The primary indicator of quality of a labeling scheme is the size of the labels that it produces, and an abundance of literature therefore discusses lower and upper bounds for the worst-case sizes of labels produced by labeling schemes.

This thesis introduces the general concept of labeling schemes and gives an overview of the best known lower and upper bounds for five different types of labeling schemes and a selection of families of trees. The results presented are either "state of the art" from the literature or novel contributions of the thesis.



Contents



Preface

This text constitutes my master thesis for the Master of Science (cand.scient.) degree in computer science at the University of Copenhagen. It is the result of work done in the first half of 2013 under the supervision of Stephen Alstrup and Christian Wulff-Nilsen.

The thesis is a natural extension of a project [23] I completed with Stephen Alstrup as advisor. The project analyzed the nearest common ancestor (NCA) labeling scheme for trees in [6], and the analysis led to some improvements that reduced the worst-case label size of the scheme significantly. This thesis contains further improvements of these ideas (Theorems 5.11 and 5.12), extensions to other families of trees (Theorems 5.13 and 5.14) as well as a many other results for five different types of labeling schemes and a selection of families of trees. Some of the results from this thesis have already been combined into an article and submitted to a conference, and others will follow shortly—see below for further detail.

The reader is assumed knowledgeable about the basics of sets, graphs, trees, strings and complexity theory. Nevertheless, to assist the reader, the preliminaries outline basic definitions, notation and terms. The remainder of the thesis is structured as follows.

Chapter 1 is a general introduction to labeling schemes and labeling problems. In addition to presenting some general definitions, the chapter outlines the overall goals of the thesis and discusses how and what results are presented throughout. Tables 1.1 and 1.2 at the end of the chapter give an overview of all the results presented in the thesis as well as of the current status of the labeling problems under consideration.

Chapters 2 to 6 are where all the fun happens. These chapters discuss the labeling problems for adjacency, ancestry, sibling, NCA and distance and present lower and upper bounds on worst-case label sizes for a selection of tree families. The results presented are "state of the art", at least to the best knowledge of me and my advisors, meaning that no (significantly) tighter lower and upper bound on worst-case label sizes are known to any of us. Extra attention has been given to the NCA and distance problems for trees, binary trees and caterpillars, since the thesis contains novel contributions in these areas.

Finally, Chapter 7 contains a few concluding remarks and outlines possible

directions for future studies.

In addition to the main chapters, the thesis has two appendices. Appendix A contains a small selection of results that are not specific to labeling schemes but can be applied in a variety of contexts. These results have been placed together to facilitate referencing and allow re-use and have been removed from the main text to avoid cluttering up the presentation of labeling schemes with technical details. Appendix B contains a small discussion of some of the ambiguities in the existing literature concerning which parameters are known to the decoder in a labeling scheme.

The results in the thesis originate from three sources: some can be found "as is" in the literature; some easily follow from a published result or can be derived using a similar technique; and some are completely new. In the first two cases, the relevant publications have been cited. If no external source has been cited for a nontrivial result, it means that the result is new and a contribution of the thesis.

The main contributions of the thesis are:[1]

(1) An improved upper bound for the NCA labeling problem for trees (Theorem 5.11). The lowest upper bound from the literature is $10 \log n + O(1)$ from [6], which was improved to $4\lfloor \log n \rfloor$ in [23], and here is improved to $(1 + \log(2 + \sqrt{2})) \log n + O(1) \approx 2.772 \log n + O(1)$.

(2) Upper bounds for the NCA labeling problem for binary trees and caterpillars (Theorems 5.13 and 5.14).

(3) A writeup and some improvements of an unpublished proof sketch by Alstrup and Larsen [8] of a lower bound for the NCA labeling problem for trees (Theorem 5.4). Their proof establishes the lower bound $1.001 \log n - O(1)$ which here is improved to $1.008 \log n - O(1)$.

(4) A fix of some flaws in a proof by Gavoille et al. [21] of a lower bound of $\frac{1}{8} \log^2 n - O(\log n)$ for the distance labeling problem for trees (Theorem 6.5).

(5) A (nontrivial) lower bound of $2 \log n - \log \log n - O(1)$ for the distance labeling problem for caterpillars (Theorem 6.6). The matching upper bound is $2 \log n$, so this shows that the scheme is optimal in the dominant term.

(6) An improved upper bound for the distance labeling problem for trees (Theorem 6.8) established by extending an NCA labeling scheme. The new upper bound is $\frac{1}{2} \log^2 n + O(\log n)$, which improves the constant factor of [21] from $1/(\log 3 - 1) \approx 1.710$ to $1/2$ and hence reduces the gap to the lower bound.

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

(7) An elegant proof of a known upper bound for the adjacency labeling problem for general graphs (Theorem 2.3).

(8) Numerous other lower and upper bounds for special cases (binary trees, caterpillars, trees of bounded depth and trees with bounded maximum degree) for all five labeling problems. Most of these have not been considered previously.

(9) A general discussion of some of the ambiguities in the existing literature concerning which parameters are known to the decoder in a labeling scheme (Appendix B), and a less ambiguous notation (introduced in Chapter 1).

The contributions in (1)–(3) have been combined into an article [7] that has been submitted to the 2014 ACM-SIAM Symposium on Discrete Algorithms (SODA). Likewise, the contributions in (4)–(6) will soon be combined into an article that will be submitted to a leading conference. In addition, the majority of results in the thesis will contribute to a survey on labeling schemes that will be submitted to a peer-reviewed journal.

I would like to thank Stephen Alstrup, Christian Wulff-Nilsen and Noy Rotbart for many inspiring, interesting and sometimes even amusing discussions. I am also grateful to Ron Benner and all the people who were working at his lab during my visit to the University of South Carolina from March to May 2013 for warmly welcoming me and accepting me as one of their own even though my field of study was somewhat remote from theirs. Finally, I am deeply indebted to my fiancée, Linda Jørgensen, for always supporting me, believing in me and putting the joy into everything I do.

Esben Bistrup Halvorsen
Copenhagen, July 2013

Chapter 0

Preliminaries

The reader is assumed knowledgeable about the basics of set theory, information theory, graph theory and complexity analysis. For convenience, this preliminary chapter covers notation and terminology used throughout the thesis. Most of the material below can be skipped for now if one is familiar with the aforementioned subjects and their standard notation and terminology.

0.1 Numbers and sets

The set of real numbers is denoted $\mathbb{R}$, the subset of integers is denoted $\mathbb{Z}$ and the subset of (strictly positive) natural numbers is denoted $\mathbb{N}$. The extension of $\mathbb{N}$ that includes 0 is denoted $\mathbb{N}_0$.

Given a real number x, we denote by $\lfloor x \rfloor$ the largest integer that is smaller than or equal to x and by $\lceil x \rceil$ the smallest integer that is larger than or equal to x. The logarithm in base b of $x > 0$ is denoted $\log_b x$. When $b = 2$ we simply write $\log x$. The *iterated logarithm* $\log^* x$ of x is the number of times one needs to apply log to x in order to get a number less than or equal to 1. More formally, $\log^* x = 0$ for $0 < x \leq 1$ and, recursively, $\log^* x = 1 + \log^*(\log x)$ for $x > 1$.

We can represent a number $n \in \mathbb{N}$ in base $b \in \mathbb{N}$ by writing it as a sequence $d_k \cdots d_0$ for integers $d_0, \ldots, d_k$ with $0 \leq d_i < b$ and $d_k \neq 0$ such that $n = \sum_{i=0}^{k} d_i b^i$. The integers d_i are called *digits* and we do not separate digits with commas but always make sure to use distinct symbols for the b possible values of a digit (see the next section). The number of digits in this representation is $k + 1 = \lfloor \log_b n \rfloor + 1$. The special case of $n = 0$ can be written in base b either as the single digit 0 or, in some cases, as the empty sequence containing no digits at all. Appendix A.1.1 presents a few results on how numbers can be efficiently represented as code.

0.2 Strings

An *alphabet* Σ is a finite, non-empty set whose elements are denoted *symbols*. The set of finite sequences of elements from Σ is denoted Σ^*, and the elements of Σ^* are the *strings* over Σ. We usually write the symbols of a string next to each

other rather than separated by commas, and we use a true type font to distinguish strings from other mathematical entities: for example, 2013 is a natural number, whereas `2013` is a string over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{2}, \mathtt{3}, \mathtt{4}, \mathtt{5}, \mathtt{6}, \mathtt{7}, \mathtt{8}, \mathtt{9}\}$ which represents the number 2013 in base 10. A *binary* string is a string over the alphabet $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. The symbols in a binary string are called *bits*, including when they are used as digits to represent a number in base 2.

The *length*, or *size*, of a string $s \in \Sigma^*$ is the number of symbols in it and is denoted $|s|$. The *empty* string is the (unique) string consisting of no symbols at all. It is denoted ε and has length $|\varepsilon| = 0$

The *concatenation* of two strings $s = s_1 \cdots s_m$ and $t = t_1 \cdots t_n$ is the string $s \cdot t$ defined by $s \cdot t = s_1 \cdots s_m t_1 \cdots t_n$ and with length $|s \cdot t| = |s| + |t|$. If a string s can be written as $s = s' \cdot s''$, then s' is a *prefix* of s, and s'' is a *suffix* of s. Note, in particular, that the empty string is a prefix and a suffix of all other strings. The symbols (or digits, when the alphabet consists of numbers) of s' and s'' are the *most significant* and *least significant*, respectively, symbols (or digits) of s.

An order $\leq$ on the the alphabet Σ induces the *lexicographical order* $\leq_{\text{lex}}$ on Σ^* defined for $s, t \in \Sigma^*$ by

$$s \leq_{\text{lex}} t \iff \begin{cases} s \text{ is a prefix of } t; \\ s = s_1 \cdot s',\ t = t_1 \cdot t' \text{ and } s_1 < t_1; \text{ or} \\ s = s_1 \cdot s',\ t = s_1 \cdot t' \text{ and } s' \leq_{\text{lex}} t' \end{cases}$$

where $s_1, t_1 \in \Sigma$ and $s', t' \in \Sigma^*$. Here, we have written $x < y$ as short for $x \leq y \wedge x \neq y$, and we likewise define $s <_{\text{lex}} t$ to be short for $s \leq_{\text{lex}} t \wedge s \neq t$.

0.3 Graphs

A *graph* G consists of a finite set of *nodes* V and a set of *edges* E consisting of 2-element subsets of V. The edges represent connections between the nodes, and we write uv as short for the edge $\{u, v\}$. We also occasionally write $v \in G$ to mean $v \in V$. If u and v are nodes and $e = uv$ is an edge, then we say that e is an edge *between* u and v, that u and v are *connected by* e, that u and v are *neighbors*, or that e is an edge *from* u *to* v or vice versa. Note that there cannot be multiple edges between two nodes; that there can be no edge from a node to itself; and that edges are not directed, meaning that there is no difference between the edge uv and the edge vu. In other words, the graphs under investigation in this thesis are *simple*, *unweighted* and *undirected*.

A *subgraph* G' of G is a graph whose node set V' is a subset of V and whose edge set E' is a subset of E. Obviously, the edges in E' must be between nodes in V', but it need not consist of *all* the edges from E that are between nodes from V'. We say that G' is an *induced subgraph* of G if E' does, in fact, consist of all edges from E that are between nodes in V': that is, if $E' = \{uv \in E \mid u, v \in V'\}$.

The *degree* of a node v is the number of edges from v to other nodes and is denoted $\deg(v)$. The degree of a node is always a number between 0 and $|V|-1$. A graph has *bounded degree* Δ if all nodes in it have degrees bounded by Δ.

A sequence $v_0, \ldots, v_k$ of nodes such that $v_{i-1}v_i$ are edges for $i = 1, \ldots, n$ is a *path* in G of length k. The graph G is *connected* if every two nodes have a path between them. Any graph is the union of connected, induced subgraphs, which are the *connected components* of G. A path is a *cycle* if it starts and ends in the same node, and a cycle is *simple* if no other nodes are repeated.

The *distance* between two nodes u and v is the length $\text{dist}(u, v)$ of a shortest path between them. If there is no such path, we set $\text{dist}(u, v) = \infty$. A single node v is considered a path of length 0, and we therefore always have $\text{dist}(v, v) = 0$.

Since the same nodes can belong to several different graphs, we sometimes write V_G, E_G, $\deg_G(v)$ and $\text{dist}_G(u, v)$ to emphasize that we are working within the graph G. Note that, if H is a subgraph of G, then $\deg_H(v) \leq \deg_G(v)$ and $\text{dist}_H(u, v) \geq \text{dist}_G(u, v)$ for all $u, v \in V_H \subseteq V_G$.

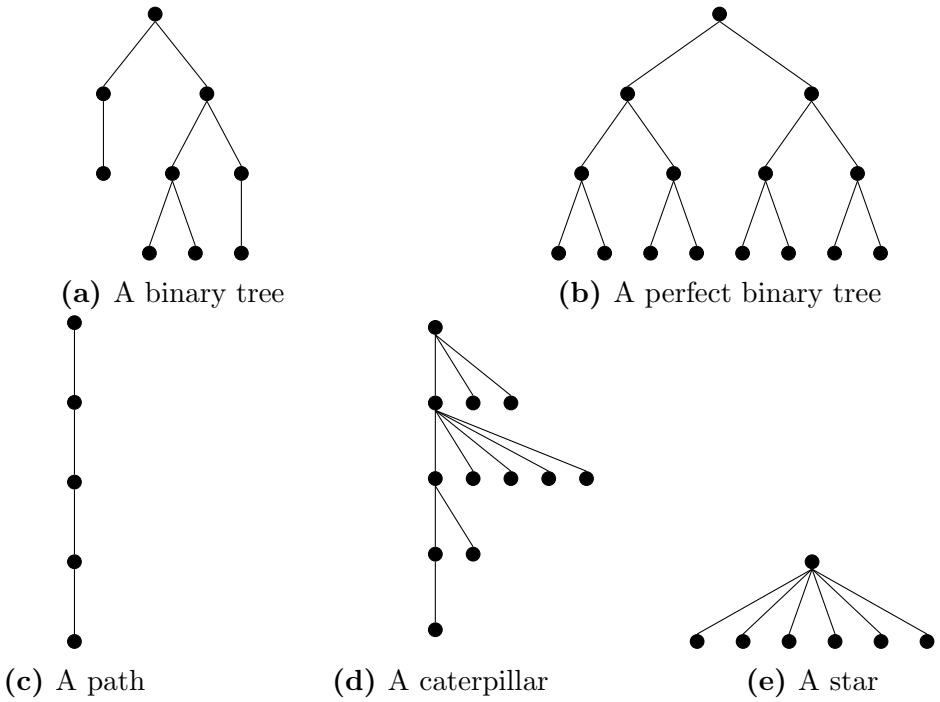
0.4 Trees

A *tree* T is a graph that is connected and contains no cycles. The number of edges in a tree is always $|E| = |V| - 1$, and every two nodes are connected by a unique path. Given three nodes, their *center* is the unique node which is connected to the three nodes with three edge-disjoint paths.

A tree may contain a specially designated node called the *root*, in which case the tree is *rooted*. The nodes of degree 1 other than the root are called *leaves*, and all other nodes are called *internal* nodes. The distance from a node v to the root is called the *depth* of v, denoted $\text{depth}(v)$. The depth of the tree, $\text{depth}(T)$, is the maximum depth among the nodes in T.

If u is a node on the path from the root of a rooted tree to a node v, then u is an *ancestor* of v, and v is a *descendant* of u. If, in addition, $\text{depth}(v) = \text{depth}(u) + 1$ so that uv is an edge, then u is the unique *parent* of v, denoted $\text{parent}(v)$, and v is a *child* of u. Two nodes that have the same parent are *siblings*. Note, in particular, that a node is its own ancestor, descendant and sibling, but not its own child or parent. A *common ancestor* of two nodes is a node that is an ancestor of both nodes, and their *nearest common ancestor* (NCA) is the unique common ancestor with maximum depth. Given a node v, the descendants of v form an induced subtree T_v with v as root. The *size* of v, $\text{size}(v)$, is the number of nodes in T_v: that is, the number of descendants of v.

A *binary tree* is a rooted tree in which all nodes have at most two children. A binary tree T is *full* if all internal nodes have exactly two children and is *complete* if there are exactly 2^i nodes with depth i for all $i < d$, where d is the depth of the tree. A binary tree that is both full and complete and where all leaves are at the same depth is *perfect*. A perfect binary tree will have 2^i nodes of depth i for



(a) A binary tree **(b)** A perfect binary tree

(c) A path **(d)** A caterpillar **(e)** A star

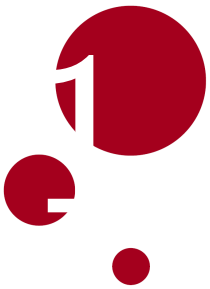
Figure 1: How to recognize different types of trees from quite a long way away.

all $i \leq d$ and a total of $2^{d+1} - 1$ nodes, whereof 2^d are leaves.

A *caterpillar* is a tree in which all leaves are connected to a single path, denoted the *main path*. A *star* is a tree in which all nodes but one are leaves.

0.5 Complexity

The notation and terminology used in the thesis is completely standard. The reader is referred to [14] for further detail.



Chapter 1

Labeling schemes

1.1 Introduction

Graphs play an important role in modeling a huge variety of structures: street maps, the Internet, biological classifications, social networks, XML documents, pedigree charts, company organizations etc. Traditional representations of such structures may require access to a global data structure in order to retrieve information, even if the desired information pertains only to a small subset of the structure. In contrast, a localized or distributed representation is based on breaking down the global structure into small, locally stored pieces such that one can infer information directly from the local data.

Graph *labeling schemes* are based on the idea of building information directly into the nodes of a graph. Given a graph, a labeling scheme associates with each node a *label*, which is a piece of data, usually in the form of a binary string, such that any subsequent queries regarding properties of a small subset of the graph can be answered by inspecting the labels of the nodes in the subset without accessing any additional sources of information.

Labeling schemes, as we define them in this thesis, are built for specific types of queries. An *adjacency* labeling scheme, for example, labels each node in a graph in such a way that the adjacency between two nodes can be determined directly from the labels of the nodes. Other types of labeling schemes may be built for different types of queries, for example queries regarding ancestry, distance, routing, connectivity, nearest common ancestors and siblings.

Labeling schemes for adjacency queries were the first to be investigated in the literature. A restricted form of adjacency labeling schemes for graphs were studied almost 50 years ago by Breuer and Folkman [12, 13]. The more general concept of was defined 25 years later by Kannan, Naor and Rudich [27] as well as by Müller [35]. Following that, the idea lay dormant for over a decade until it was noticed that labeling schemes could also be defined for many other types of queries other than adjacency [37, 21, 30, 28, 2, 9, 29, 1, 18, 40, 39, 6, 5, 31]. This observation revived interest in labeling schemes and triggered an abundance of subsequent publications, including many variants of the concept (see Section 1.6 for examples). A formal and more general definition of *informative labeling schemes* was given in [36]. Further detail on the historical development

of labeling schemes up until 2001 can be found in [20].

If the query type supported by a labeling scheme allows the entire graph to be reconstructed from the labels, then collection of labels can be seen as an *implicit representation* of the graph: that is, a representation where the graph is not stored as a single structure but can be determined from a collection of smaller structures. This is in contrast to any global representation of a graph, for example an adjacency matrix, where the adjacency between two nodes is determined by consulting the relevant entry in a single entity, and where the index of each node contains no relevant information in itself but serves only as a placeholder or a pointer to the entries of the matrix. In this sense, a labeling scheme is more efficient since it does not waste space on meaningless placeholders. This, however, does not mean that the collection of labels in total uses less space than a traditional representation: on the contrary, the extra requirement that the data must be distributed may lead to a data structure that is larger in total.

It is not hard to create labeling schemes for any type of query if labels are unrestricted in size. One of the main concerns of a labeling scheme, and one of the biggest motivations for new results on the subject, is therefore to produce labels that are as small as possible. The ultimate goal is to find a labeling scheme whose labels cannot be smaller in the worst-case scenario. This problem can be attacked from two sides: from above by proving an upper bound on label sizes (typically be presenting a labeling scheme with that particular label size); and from below by proving a lower bound.

There are many practical applications of labeling schemes. As an example, consider Extensible Markup Language (XML) documents [1], which are becoming an increasingly popular and ubiquitous standard for exchanging structured data on the Internet. An XML document can be viewed as a rooted tree in which each node corresponds to a semantic element, enclosed by matching begin and end tags in the form `<item>...</item>`. When searching for information in an XML document, one will typically not only search for pure text but also utilize the semantic structure of the document and specify, for example, specific ancestry relations in the document tree. By using a labeling scheme, queries of this type can be answered directly from labels, which can be stored in a hash table, without having to access the actual document. This can have a significant positive impact on performance.

To achieve a good performance in the XML example, it is important that as large a part of the index structure can reside in main memory. Since these structures can be extremely large, every single bit counts, so the hunt for labeling schemes with small labels is not only of theoretical interest. Nonetheless, only very few labeling schemes have been proven to be optimal in the sense that their worst-case label sizes are as small as theoretically possible. The field of labeling schemes is still very young and wide open for new results.

1.2 An example

Before diving into the details of the precise definition of labeling schemes, it is instructive to warm up with a small example accompanied by some general remarks. This section therefore presents a simple *adjacency labeling scheme for trees*: what this means will be made precise later, but the general idea is to construct an algorithm that associates with every node a *label*, which is just some data, such that, given the labels of two nodes, one can determine if the nodes are adjacent or not. The labeling scheme presented here is a variant of the original example by Kannan, Naor and Rudich [27].

Consider an arbitrary rooted tree with n nodes. Enumerate the nodes in the tree with the numbers 0 through $n-1$, and let, for each node v, $l(v)$ be the number associated with v. Now, give the root node r the label $(l(r), l(r))$ and every other node v the label $(l(u), l(v))$, where u is the unique parent of v. In practice we need to encode these labels as binary strings, but we ignore this technicality for now. Given the labels $(l(u), l(v))$ and $(l(u'), l(v'))$ for two nodes v and v', we can now determine if v and v' are adjacent exclusively by inspecting the labels: v and v' are adjacent if and only if either $l(u) = l(v')$ or $l(v) = l(u')$ but not both. (The exclusive "or" is needed to prevent the root from being classified as adjacent to itself.)

This is a labeling scheme. It consists of an *encoder* algorithm that labels the nodes of a tree and a *decoder* algorithm that can answer adjacency queries using only labels as input. Note that the encoding algorithm relies on knowing the entire tree, whereas the decoding algorithm only knows the labels it receives as input and nothing else. This is the whole point of a labeling scheme: information is stored locally at each node, and subsequent queries (in this case concerning adjacency) can therefore be distributed to the individual nodes.

One could, of course, store the entire tree at each node, but this would defy the aim of distributing information rather than just replicating it. Therefore, the primary indicator of the quality of a labeling scheme is the size of the labels it produces in the worst-case scenario. In the above example, the pair $(l(u), l(v))$ can be represented as a binary string of length $2\lceil \log n \rceil$. Can we do better? That is, can we find another labeling scheme which produces even smaller labels in the worst-case scenario? If yes, how does it look? If no, why not? These are some of the main questions asked in this thesis. The ultimate goal is to find an *optimal* labeling scheme: that is, one where the worst-case label size is as small as theoretically possible.

In many cases, the worst-case label size for an optimal labeling scheme remains unknown, and we have to settle for upper and lower bounds for it. The very existence of the labeling scheme presented here means that the optimal worst-case label size is upper bounded by $2\lceil \log n \rceil$. As we shall see, there does, in fact, exist an adjacency labeling scheme with an even smaller worst-case label size of $\log n + O(\log^* n)$. It is also not hard to get a lower bound: a path with n nodes

must have uniquely labeled nodes, since no two nodes have the same adjacency relations, and hence we need at least n distinct labels, which means that there must exist a label of size at least $\lfloor \log n \rfloor$. So the optimal worst-case label size is somewhere between $\lfloor \log n \rfloor$ and $\log n + O(\log^* n)$. What it actually is remains unknown—there is still work left to be done.

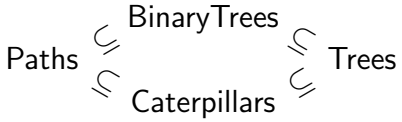
There are two main characteristics of a labeling scheme. First, there is the type of query that the labeling scheme has been constructed for. In our example, it is "adjacency", but it could also have been something else, for example "ancestry" (is one node an ancestor of the other?) or "distance" (what is the distance between two nodes?). Second, there is the family of graphs under consideration. In our example, the labeling scheme is for the family of all trees, but both smaller and larger families would have been possible. A different choice of query or a different choice of graph family may lead to an entirely different labeling scheme with different properties and a different optimal worst-case label size. Section 1.4 presents an exact definition of "f-labeling schemes for $\mathcal{G}$", where f represents the type of query and $\mathcal{G}$ is the graph family under consideration.

1.3 Graph families

This section introduces some standard notation for *families* (or *categories*) of graphs[1] that will be used throughout the thesis. We begin with some main families:

Graphs: the family of graphs,
Trees: the family of rooted trees,
BinaryTrees: the family of (rooted) binary trees,
Caterpillars: the family of rooted caterpillars,
Paths: the family of rooted paths.

Binary trees come with a natural rooting, and we assume a path to always be rooted at one of its leaves and a caterpillar to always be rooted at one of the end nodes of its main path. Since a path is both a caterpillar and a binary tree, which are both trees, we have



The family Paths is not very interesting on its own, but it is useful to have for the very reason that it is small enough to be contained in all the other families.

[1] See Sections 0.3 and 0.4 in the preliminaries for definitions and notation for different types of graphs and trees.

In some situations we do not actually need our trees to be rooted, but in these cases we can just ignore the roots. When doing so, we also have the inclusion $\mathsf{Trees} \subseteq \mathsf{Graphs}$.

From these main families, we derive a multitude of subfamilies by specifying upper bounds on a few invariants. To formalize this, we will abuse notation slightly and make sure always to denote bounds on the number of nodes, the maximum degree and the depth for an *entire* family of graphs by N, Δ and D, respectively, whereas we shall use n, δ and d for the number of nodes, the maximum degree and the depth of a *specific* graph. For a family $\mathcal{G}$ of graphs, we can now define the following:

$$\begin{aligned}
\mathcal{G}(N) &= \text{the subfamily of } \mathcal{G} \text{ of graphs with } n \leq N \text{ nodes,} \\
\mathcal{G}(\Delta) &= \text{the subfamily of } \mathcal{G} \text{ of graphs with maximum degree } \delta \leq \Delta, \\
\mathcal{G}(D) &= \text{the subfamily of } \mathcal{G} \text{ of trees with depth } d \leq D.
\end{aligned}$$

The latter is only defined for $\mathcal{G} \subseteq \mathsf{Trees}$. Surely, these definitions would cause confusion if we replaced N, Δ or D by concrete values, but this will not happen in this thesis. In any case, we could always write, for example, "$\mathcal{G}(\Delta = 3)$" or "$\mathcal{G}(\Delta)$ for $\Delta = 3$" rather than the ambiguous "$\mathcal{G}(3)$". We will also allow combinations of the parameters N, Δ and D by defining

$$\begin{aligned}
\mathcal{G}(N, \Delta) &= \mathcal{G}(N) \cap \mathcal{G}(\Delta) \\
\mathcal{G}(N, D) &= \mathcal{G}(N) \cap \mathcal{G}(D) \\
\mathcal{G}(\Delta, D) &= \mathcal{G}(\Delta) \cap \mathcal{G}(D)
\end{aligned}$$

Thus, for example, $\mathsf{Caterpillars}(N)$ is the family of caterpillars with $n \leq N$ nodes, and $\mathsf{Trees}(\Delta, D)$ is the family of trees with maximum degree $\delta \leq \Delta$ and depth $d \leq D$. Note that $\mathsf{BinaryTrees}$ is a subfamily of $\mathsf{Trees}(\Delta)$ for $\Delta = 3$ and that Paths is a subfamily of $\mathsf{Trees}(\Delta)$ for $\Delta = 2$. (These are just inclusions and not identities because of the rooting.) Since the maximum degree of paths and binary trees are naturally bounded, we will never need to specify a Δ for these two families. Further, since any tree with n nodes, maximum degree δ and depth d satisfies $\delta + 1, d + 1 \leq n \leq \delta^{d+1} - 1$, we would normally never need to specify a bound on all three parameters at the same time.

A relevant question to ask is why we define $\mathcal{G}(N)$ to be the subfamily of graphs with *at most* N nodes rather than *exactly* N nodes. This is discussed briefly in Section 1.6.

1.4 Definition of labeling schemes

Suppose that $\mathcal{G}$ is a family of graphs. An *encoder for* $\mathcal{G}$ is a an algorithm, enc, that takes a graph G from $\mathcal{G}$ as input and produces a *label* $\mathrm{enc}_G(v) \in \{\mathtt{0}, \mathtt{1}\}^*$ for

each node v in G. We can think of the encoder as a family $\mathrm{enc} = (\mathrm{enc}_G)_{G\in\mathcal{G}}$ of functions $\mathrm{enc}_G\colon V_G \to \{0,1\}^*$, where each graph $G \in \mathcal{G}$ is labeled by the function enc_G. If each function enc_G is injective, we say that the encoder produces *unique labels*. For simplicity, when it is clear from the context what graph a node v belongs to, we simply write $\mathrm{enc}(v)$ instead of $\mathrm{enc}_G(v)$.

Suppose next that k is a positive integer and that S is a set. A *k-ary decoder into S* is an algorithm, dec, that takes a k-tuple of labels (produced by some specific encoder from k nodes in the same graph) as input and returns an element of S. We can think of the decoder as a partial function $\mathrm{dec}\colon (\{0,1\}^*)^k \to S$, where the word "partial" indicates that the decoder need only be defined on the subset $\{(\mathrm{enc}(v_1),\ldots,\mathrm{enc}(v_k)) \mid v_1,\ldots,v_k \in V_G,\ G \in \mathcal{G}\}$ of k-tuples of labels produced by a specific encoder, enc, from k nodes belonging to the same graph $G \in \mathcal{G}$.

Now, suppose that f is a function on k-tuples of nodes from the same graph $G \in \mathcal{G}$ to a set S. For example, f could be an adjacency function, taking a pair of nodes (u,v) from a graph $G \in \mathcal{G}$ as input and returning true if u and v are adjacent and false otherwise, in which case $k = 2$ and $S = \{\mathsf{true},\mathsf{false}\}$; or f could be a distance function, taking a pair of nodes (u,v) from a graph $G \in \mathcal{G}$ as input and returning their distance, in which case $k = 2$ and $S = \mathbb{N}_0 \cup \{\infty\}$. We can think of f as a function family $f = (f_G)_{G\in\mathcal{G}}$, where $f_G\colon V_G^k \to S$, but as long as it is clear from the context what graph a k-tuple of nodes belongs to, we do not need to distinguish between the function family and each individual function, and we simply write $f(v_1,\ldots,v_k)$ instead of $f_G(v_1,\ldots,v_k)$. An *f-labeling scheme* is a pair $\sigma = (\mathrm{enc},\mathrm{dec})$ consisting of an encoder for $\mathcal{G}$ and a k-ary decoder into S such that, for any $G \in \mathcal{G}$ and any k nodes $v_1,\ldots,v_k \in G$,

$$\mathrm{dec}(\mathrm{enc}(v_1),\ldots,\mathrm{enc}(v_k)) = f(v_1,\ldots,v_k). \tag{1.1}$$

The decoder need only be defined on k-tuples in the above form. The formula says that $f(v_1,\ldots,v_k)$ can be computed from the labels for $v_1,\ldots,v_k$ rather than from the actual nodes. Note that the encoding of a node depends on the graph to which the node belongs, whereas the decoding of a k-tuple of labels is oblivious to the graph from which the labels come.

The time spent by the encoder on labeling the nodes of a given graph is called the *encoding time*, and the time spent by the decoder to transform a k-tuple of binary strings into an element of S is called the *decoding time*. Although these time complexities are important, this thesis is only concerned with *label sizes*, which are the number of bits in the binary strings produced by the encoder. Thus, time complexities are generally ignored throughout the thesis, but in the few cases where they are mentioned, it is under the assumption of the RAM model of computation with the length of a machine word set to $\Omega(\log N)$ bits, where N is an upper bound on the number of nodes in the graphs under consideration, meaning that the basic operations on size $O(\log N)$ labels can be performed in constant time.

1.5 Labeling problems

This thesis is devoted to finding f-labeling schemes where the worst-case label size is as small as possible. In other words, the goal for a specific f and a specific $\mathcal{G}$ is to find an f-labeling scheme $\sigma = (\text{enc}, \text{dec})$ such that

$$L = \max_{G \in \mathcal{G}} \ell_G, \quad \text{where} \quad \ell_G = \max_{v \in G} |\,\text{enc}(v)|, \tag{1.2}$$

is minimal. We refer to this as the *f-labeling problem* or, if it is necessary to specify the underlying graph family, the *f-labeling problem for $\mathcal{G}$*. A labeling scheme that solves the problem is *optimal*. Note that there can be many optimal f-labeling schemes for $\mathcal{G}$.

Formula (1.2) uses ℓ_G to denote the worst-case label size in a specific graph G, whereas L is the overall worst-case label size for the entire family. If it is clear from the context (or unimportant) what the specific graph in question is, we shall simply write ℓ as short for ℓ_G. It may happen that L is not defined, because the values ℓ_G are unbounded. In such a situation, *any* labeling scheme is optimal, which makes the definition rather uninteresting. It is still possible, however, to assess the quality of a labeling scheme by describing label sizes in terms of "n", the number of nodes in a *specific* graph, or by considering the restriction of the labeling scheme to $\mathcal{G}(N)$ and describing the (asymptotic behavior of the) worst-case label size as a function of N. We shall do the former when considering *specific* labeling schemes in connection with upper bounds, and the latter when considering lower bounds. Section 1.7 gives a more detailed explanation of how results are presented throughout this thesis.

The literature contains many examples of label sizes that are presented in terms of a variable "n", but without specifying if the variable pertains to a *specific* graph or is an upper bound for a whole *family* of graphs. This thesis avoids ambiguities by using n and N, as described previously, to distinguish the two cases. Appendix B contributes to the field of labeling schemes with a general discussion of how the different interpretations are interrelated.

In most cases, no exact results are known for the f-labeling problem for $\mathcal{G}$, and we present instead upper and lower bounds for the worst-case label sizes. To get an upper bound x, we need to prove that there *exists* an f-labeling scheme for $\mathcal{G}$ with worst-case label size of *at most* x. Conversely, to get a lower bound y, we need to prove that *any* f-labeling schemes for $\mathcal{G}$ has worst-case label size of *at least* y. If $x = y$, then the upper and lower bounds match, and we have solved the f-labeling problem for $\mathcal{G}$: the labeling scheme used for the upper bound is optimal. In practice, a less strict definition of "optimal" often suffices, since we are mostly concerned with asymptotic results. Therefore, label sizes are often presented with an exact leading term followed by a term in asymptotic notation (big O, big Ω or big Θ).

An f-labeling scheme $\sigma = (\text{enc}, \text{dec})$ for $\mathcal{G}$ naturally induces an f-labeling scheme σ' for $\mathcal{G}'$ for any $\mathcal{G}' \subseteq \mathcal{G}$ by simply restricting the input sets of f, enc and

dec. We refer to σ' as the *restriction* of σ to $\mathcal{G}'$. Since the maximum in (1.2) becomes smaller when restricting $\mathcal{G}$ to $\mathcal{G}'$, the worst-case label size for σ' will be bounded by the worst-case label size for σ. This immediately implies the following important result:

Theorem 1.1. *If $\mathcal{G}' \subseteq \mathcal{G}$ are graph families and L', L are the worst-case label sizes of an optimal f-labeling scheme for $\mathcal{G}'$ and $\mathcal{G}$, respectively, then $L' \leq L$. In particular, any lower bound for L' is also a lower bound for L, and any upper bound for L is also an upper bound for L'.*

We categorize labeling schemes and labeling problems according to the function f and name them thereafter. For example, if f is an adjacency function, we use the terms *adjacency labeling scheme* and *adjacency labeling problem*. The following overview presents the five labeling problems that are discussed in this thesis. Each problem is superficially described by specifying the input and output of the function f. Chapters 2 to 6 go into the details of each individual problem.

Adjacency (Chapter 2) Given a pair of nodes in a graph, return true if the corresponding nodes are adjacent and false otherwise.

Ancestry (Chapter 3) Given a pair of nodes in a rooted tree, return true if the first node is an ancestor of the second node and false otherwise.

Sibling (Chapter 4) Given a pair of nodes in a rooted tree, return true if the nodes are siblings and false otherwise.

Nearest common ancestor (NCA) (Chapter 5) Given a pair of nodes in a rooted tree, return the label of their nearest common ancestor.

Distance (Chapter 6) Given a pair of nodes in a graph, return the distance between the nodes.

These five labeling problems are highly interconnected. In particular, since two nodes are adjacent exactly when their distance is 1, any distance labeling scheme can be changed into an adjacency labeling scheme without modifying the encoder and hence without changing the labels. Likewise, since a node is the ancestor of another node exactly when it is the NCA of both nodes, any NCA labeling scheme can be changed into an ancestry labeling scheme in the same way. In this sense, we have, informally,

$$\text{distance} \implies \text{adjacency} \qquad \text{and} \qquad \text{NCA} \implies \text{ancestry}$$

Although sibling labeling schemes in terms of label sizes are simpler than any of the above, it is generally not possible to convert any of the four other labeling schemes into a sibling labeling scheme without changing the encoder. However, if a labeling scheme is strong enough to answer both adjacency and NCA queries,

then it can also answer sibling queries, because two nodes are siblings exactly when they are both adjacent to their NCA. The same holds for a labeling scheme that is strong enough to answer both ancestry and distance queries, because two nodes are siblings exactly when their distance is 2 and neither is an ancestor of the other. Thus, we have, informally,

$$\begin{array}{c}\text{adjacency}\\ +\\ \text{NCA}\end{array} \Longrightarrow \text{sibling} \Longleftarrow \begin{array}{c}\text{ancestry}\\ +\\ \text{distance}\end{array}$$

Labeling schemes that are capable of handling multiple query types are just one of many variations that are not considered in this thesis. Section 1.6 presents some of the many different variants of the general concept of labeling schemes.

1.6 Variations

This thesis investigates five labeling problems, namely adjacency, ancestry, sibling, NCA and distance as described in Section 1.5, for a selection of graph families, namely Graphs, Trees, Trees(N), Trees(Δ), Trees(D), BinaryTrees and Caterpillars as defined in Section 1.3. We could have chosen to investigate other labeling problems, other graph families and, in fact, numerous other variations of all the concepts introduced so far. This section contains a brief discussion of why we have made these choices and what other possibilities there are.

The five labeling problems are, in our opinion, important when it comes to their historical role, their applicability to real life problems and their theoretical properties. The selection is also "broad": two of the five problems are defined for general graphs whereas the other three only are defined for trees; three of the five labeling problems are based on Boolean queries, whereas NCA and distance queries return binary strings and numbers, respectively; and several of the problems come in unique and a non-unique variants (and we shall discuss both in detail for the sibling labeling problem). That said, many other labeling problems are equally interesting and varied and could easily have been our choice instead. Time and space constraints for this thesis have been the main influences on the limited selection of problems.

The main reason for our choice of graph families is that these families are well-known, easily defined and have already been studied in the literature for the selected labeling problems. Our main focus in this thesis is on trees, and we have mainly included graphs to complete the discussions of adjacency and distance. There are certainly many other natural graph families that we could have chosen, but our selection is sufficiently interspersed, from small to big, to provide valuable insight into the nature of each of the labeling problems. Indeed, even if we are interested in another family of graphs, the result in Theorem 1.1 means that we can still obtain useful information from results on smaller or bigger families.

A natural question to ask is why we define $\mathcal{G}(N)$ to be the graphs from GGG with *at most* N nodes rather than *exactly* N nodes (and similarly for Δ and D). Again, the answer is tied to what seems "natural", although opinions may vary quite a lot. As it turns out, however, it rarely makes a difference in practice if we choose one or the other: a labeling scheme that works for graphs with exactly N nodes is, in general, likely to also work for graphs with fewer than N nodes without producing larger worst-case labels. This thesis uses the "at most" definition, since it seems more likely that the user of a labeling scheme knows only upper bounds and not exact sizes for the inputs to the scheme.

In addition to the choice of labeling problem and graph family, there is a vast number of variations to the definition of labeling schemes. We shall only discuss a few of these variations in this thesis, but to give an idea of the broadness of the field, we here name some of the important variations and give a few examples. See [20] for references.

Other problems As mentioned, there are other labeling problems than the five under investigation in this thesis. Instead of considering adjacency, ancestry, sibling, NCA or distance, one could, for example, consider *connectivity* (given two nodes in a graph, return `true` if the nodes belong to the same connected component and `false` otherwise), *routing* (given two nodes in a tree, return the port number (a predefined labeling of edges at each node) for the first edge on the path from the first to the second node) or *center* (given three nodes in a tree, return their center[2]).

Other graph families A huge variety of other more or less "natural" families of graphs and trees are available, for example graphs with bounded girth, circumference, diameter, radius, density, arboricity, etc.; graphs that are k-connected, k-edge-connected, k-partite, k-variegated, etc.; planar graphs, cographs, split graphs, interval graphs, circle graphs, forests, lobsters etc.; see [15, 42] for definitions. The list goes on.

Different types of graphs The basic definitions of graphs and trees used in this thesis (and defined in the preliminaries) are simple. More advanced definitions involve edges that are directed, multiple, weighted etc. Most of the labeling problems get a completely new meaning when these basic definitions are changed.

Non-unique labels Some labeling schemes produce labels that for natural reasons must be unique for each graph. However, not all labeling problems have this natural uniqueness and hence come in two flavors: one where labels are required to be unique, and one where they are not. The sibling labeling problem, for example, is significantly easier to solve in the non-unique case. This is discussed further in Chapter 4.

[2]See Section 0.4 in the preliminaries for a definition.

Approximative Formula (1.1) requires the function f to be computed exactly from the labels of the nodes. Another option is to allow f to be computed *approximately*. This makes sense only for labeling schemes whose outputs can be approximated in some way, for example distance labeling schemes whose outputs are numbers. Approximate labeling schemes will produce labels of varying lengths, depending on the required precision of the output.

Probabilistic Instead of requiring (1.1) to always be correct, we can allow it to only be correct with a certain probability. Such probabilistic labeling schemes will produce labels of varying lengths, depending on the required likeliness of a correct answer. A variant of this are *one-sided* probabilistic labeling schemes, where only positive (or negative) answers can be correct with a probability smaller than 1.

Dynamic The graphs considered in this thesis are all static in the sense that they do not change along the way. A more realistic model is to allow dynamic updates of the graph, for example adding or removing nodes or edges or changing directions or weights of edges. Labeling schemes for this dynamic model must include algorithms for how to update labels when the graph changes.

Average and total This thesis seeks to minimize label sizes in the worst-case scenario. Another option is to minimize the average label size for each graph or the sum of all label sizes for each graph (which is equivalent when the number of nodes is fixed).

Alternative concepts Labeling problems depend on the definitions of graph theoretical concepts such as "sibling" or "ancestor". If these are altered, the associated problem changes. For example, if a node is no longer defined to be its own ancestor, it will suddenly be possible to use non-unique labels for the ancestor labeling problem.

Predefined labels An interesting variant of the NCA and routing labeling problems allows nodes to have predefined labels or port numbers. In this case, the task of the labeling scheme is to produce the predefined NCA labels or port numbers using the (new) labels of two nodes.

Label lookup A variant of labeling schemes allows the decoder to look up labels of additional nodes after it receives its input labels. This makes it easier to keep label sizes at a minimum and, in particular, makes it easier to create labeling schemes with predefined labels.

Partial labeling Instead of requiring all nodes in a graph to be labeled, one could define a partial labeling scheme that only labels some of the nodes.

A *leaf* labeling scheme, for example, would only label the leaves in a graph and would only require queries to be answered for leaf labels.

Multiple query types Instead of defining labeling schemes to encode/decode a single function f, one could define $(f_1, \ldots, f_t)$-labeling schemes to be labeling schemes that support queries for a whole family of functions $f_1, \ldots, f_t$. Thus, for example, one could consider (adjacency, NCA)-labeling schemes.

1.7 The results presented in this thesis

This thesis presents a selection of results for a selection of labeling problems for a selection of families of graphs and, in particular, trees. The ultimate goal is to present "state of the art" lower and upper bounds on worst-case label sizes for each combination of labeling problem and graph family. Because of Theorem 1.1 this multitude of results can often be derived from a much smaller number of results. Each chapter therefore only presents lower and upper bound theorems for a few graph families, leaving it to the reader to derive results for the desired graph families using Theorem 1.1.

The theorems generally present results that are as strong as possible, but in many cases a few bits have sacrificed here and there in order to simplify formulas and proofs as much as possible. Some combinations of problem and graph family have certainly received more attention than others in the exposition. Quite a few pages are devoted to the NCA and distance problems for trees, binary trees and caterpillars, which is due to the fact that the thesis contains some novel contributions for these problems. This extra focus on certain areas means that there may still be many "low-hanging fruits" left to pick in other areas; see Chapter 7 for perspectives on future research.

Two tables are presented in each chapter. The first table gives an overview of the results presented in the chapter, including references to the corresponding theorems as well as to any relevant literature. The second table presents some of the consequences when applying Theorem 1.1 to these results in order to obtain lower and upper bounds for a fixed selection of graph families. Table 1.1 on page 15 gathers all the results from the tables of the first type and hence gives an overview of all the results in the thesis. Table 1.2 on page 16 gathers all the lower and upper bounds from the tables of the second type and hence gives a comparative overview of the current status of each labeling problem. Some theorems have extra assumptions on the sizes of N, Δ or D, but such assumptions are ignored when presenting results in the tables, since all but one of them are insignificant. The single exception is the lower bound for the NCA labeling problem for trees of bounded degree (and hence also general trees) in Theorem 5.4, where N is required to be rather huge. Note that the assumption $\Delta+1, D+1 \leq N \leq \Delta^{D+1}-1$, which occurs in some theorems, is natural, since any tree with n nodes, maximum

degree δ and depth d must satisfy $\delta + 1, d + 1 \leq n \leq \delta^{d+1} - 1$.

For each theorem in the thesis, a reference to the literature is given not only when the theorem has already been published, but also when the theorem is an easy consequence of a published result or can easily be obtained using the same methods or ideas as in a publication. These citations are given in the text preceding the theorem as well as in the tables of the first type. Thus, a publication of a lower bound for a specific labeling problem for a specific graph family will be cited in theorems for *any* labeling problem and *any* graph family whose proofs are similar to or can be derived from the one in the publication. The rule of thumb is: if a theorem uses a method or an idea from a certain published result, and it seems likely that the authors behind the publication could have derived the theorem themselves without too much trouble, their original work will be cited for that theorem. Consequently, if a result has *not* been equipped with a citation, it either means that it is so trivial that no single publication can rightfully claim the rights to it, or it means that the result is an original contribution of this thesis. We leave it to the reader to distinguish these two cases.

The theorems and tables in this thesis use ℓ, n, δ and d to denote the worst-case label size, the number of nodes, the maximum degree and the depth, respectively, for a *specific* graph, whereas L, N, Δ and D are used as their global counterparts for the entire graph family in question. As we shall see, lower bound results will be presented as inequalities that relate L to N, Δ and D, whereas upper bound results will be presented as inequalities that relate ℓ to n, δ and d (and in some cases also to N, Δ or D). Appendix B contains an elaborate discussion of the rationale behind why we consider individual graphs when presenting upper bounds, and the entire graph family when presenting lower bounds. The kill some of the suspense, we here list a few of the arguments from Appendix B.

The worst-case label size for a labeling scheme for a graph family with no bounds on the number of nodes is likely to be infinite. This, in general, makes it more natural to always consider graph families with a bounded number of nodes: for example, $\mathsf{Trees}(N)$ rather than Trees. For a specific labeling scheme, however, it is often possible to describe the worst-case label size in terms of an individual graph rather than for the entire graph family as a whole. Therefore, an upper bound on worst-case label sizes for a specific labeling scheme can be presented as, say, $3\lfloor \log n \rfloor$, where n is the number of nodes in the individual graph being labeled. Something similar is not sensible for lower bounds, since a lower bound for a specific graph is often too weak to offer any real insight into the overall worst-case label size for the entire family.

We prefer to present upper bounds in terms of n rather than N, since it is more precise and yields stronger results. In addition, we can, by doing so, even describe worst-case label sizes for individual graphs in a graph family $\mathcal{G}$ where the number of nodes is unbounded and where the overall worst-case label size is infinite. As it turns out these upper bounds are often the same as those for $\mathcal{G}(N)$, since the encoder and decoder in a labeling schemes rarely need to "know"

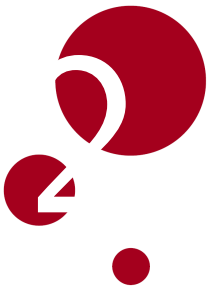
N in order to work, and hence we can obtain stronger results by considering the unbounded case for upper bounds. If one prefers to have an upper bound that does not pertain to a specific graph, this is completely trivial to obtain from the stronger result: for example, an upper bound of $3\lfloor \log n \rfloor$ for Trees immediately implies an upper bound of $3\lfloor \log N \rfloor$ for $\mathsf{Trees}(N)$.

	Family	Variant	Bound on label size	Reference
Adjacency	Graphs(N)		$L \geq \lfloor \frac{1}{2}N \rfloor - 1$	Theorem 2.1, [34, 21]
	Trees(N, D)		$L \geq \lfloor \log N \rfloor$	Theorem 2.2
	Paths(N)		$L \geq \lfloor \log N \rfloor$	Theorem 2.2
	Graphs		$\ell \leq \lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$	Theorem 2.3, [34]
	Trees		$\ell \leq \log n + O(\log^* n)$	Theorem 2.4, [10]
	Trees		$\ell \leq \log n + 3\log d + 2\log\log d + O(1)$	Theorem 2.5, [17]
	Trees(N)		$\ell \leq \log N + 3\log d + O(1)$	Theorem 2.5, [17]
	Trees(D)		$\ell \leq \log n + 3\log D + O(1)$	Theorem 2.5, [17]
	Caterpillars		$\ell \leq \lceil \log n \rceil + 6$	Theorem 2.6, [11]
	BinaryTrees		$\ell \leq \log n + O(1)$	Theorem 2.7, [11]
Ancestry	Trees(N, D)		$L \geq \lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	Theorem 3.1, [5]
	BinaryTrees(N)		$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	Theorem 3.2, [5]
	Paths(N)		$L \geq \lfloor \log N \rfloor$	Theorem 3.3
	Trees		$\ell \leq \log n + 6\log\log n + O(1)$	Theorem 3.4, [18]
	Trees(N)		$\ell \leq \log N + 4\log\log N + O(1)$	Theorem 3.4, [18]
	Trees		$\ell \leq \log n + 2\log d + 2\log\log d + O(1)$	Theorem 3.5, [18]
	Trees(N)		$\ell \leq \log N + 2\log d + O(1)$	Theorem 3.5, [18]
	Trees(D)		$\ell \leq \log n + 2\log D + O(1)$	Theorem 3.5, [18]
	Caterpillars		$\ell \leq \lfloor \log n \rfloor + 1$	Theorem 3.6
Sibling	Paths(N)	NU	$L \geq \lfloor \log N \rfloor$	Theorem 4.1
	Trees(N, D)	NU	$L \geq \lfloor \log N \rfloor - 1$	Theorem 4.2
	Caterpillars(N, Δ)	U	$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor - 2$	Theorem 4.3, [5]
	Trees(D)	U	$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	Theorem 4.4, [5]
	Paths(N)	U	$L \geq \lfloor \log N \rfloor$	Theorem 4.5
	Trees	NU	$\ell \leq \lfloor \log n \rfloor$	Theorem 4.6, [5]
	Trees(Δ)	U	$\ell \leq \lfloor \log n \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$	Theorem 4.7, [5]
	BinaryTrees	U	$\ell \leq \lfloor \log n \rfloor + 1$	Theorem 4.7
	Trees(N)	U	$\ell \leq \lfloor \log N \rfloor + \lfloor \log \lfloor \log \delta \rfloor \rfloor + 1$	Theorem 4.7, [5]
	Trees	U	$\ell \leq \lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$	Theorem 4.7, [5]
	Trees	U	$\ell \leq \lfloor \log n \rfloor + 2\lceil \log \lfloor \log \delta \rfloor \rceil + 1$	Theorem 4.7, [5]
NCA	Trees(N, D)		$L \geq \lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	Theorem 5.1, [5]
	BinaryTrees(N)		$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	Theorem 5.1, [5]
	Paths(N)		$L \geq \lfloor \log N \rfloor$	Theorem 5.1
	Trees(N)		$L \geq \lfloor 1.008 \log N \rfloor - 317$	Theorem 5.4, [8, 7]
	Trees		$\ell \leq \lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$	Theorem 5.11
	BinaryTrees		$\ell \leq \lceil (1 + \log 3)\lfloor \log n \rfloor \rceil + 1$	Theorem 5.13
	Caterpillars		$\ell \leq \lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil + 1$	Theorem 5.14
Distance	Graphs(N)		$L \geq \lfloor \frac{1}{2}N \rfloor - 1$	Theorem 6.1, [34]
	Trees(N, D)		$L \geq \lfloor \log N \rfloor$	Theorem 6.2
	BinaryTrees(N)		$L \geq \lfloor \frac{1}{2} \lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	Theorem 6.5, [21]
	Caterpillars(N)		$L \geq 2\lfloor \log N \rfloor - \lfloor \log \lfloor \log N \rfloor \rfloor - 4$	Theorem 6.6
	Graphs		$\ell \leq \lfloor (\log 3)n \rfloor + \lceil \log n \rceil$	Theorem 6.7, [43]
	Trees		$\ell \leq \lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2} + 1))\lceil \log n \rceil \rceil$	Theorem 6.8
	BinaryTrees		$\ell \leq \lceil \frac{1}{2}\lceil \log n \rceil^2 + (\frac{3}{2} + \log 3)\lceil \log n \rceil \rceil + 1$	Theorem 6.9
	Caterpillars		$\ell \leq 2\lfloor \log n \rfloor$	Theorem 6.10

Table 1.1: Bounds on the optimal worst-case label sizes for the adjacency, ancestry, non-unique (NU) and unique (U) sibling, nearest common ancestor (NCA) and distance labeling problems for different families of graphs. See Section 1.7 for further detail.

	Family $\mathcal{G}$	Lower bound on L for $\mathcal{G}(N)$	Upper bound on ℓ for $\mathcal{G}$
Adjacency	Graphs	$\lfloor \frac{1}{2}N \rfloor - 1$	$\lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$
	Trees	$\lfloor \log N \rfloor$	$\log n + O(\log^* n)$
	Trees(Δ)	$\lfloor \log N \rfloor$	$\log n + O(\log^* n)$
	Trees(D)	$\lfloor \log N \rfloor$	$\log n + 3\log D + O(1)$
	BinaryTrees	$\lfloor \log N \rfloor$	$\log n + O(1)$
	Caterpillars	$\lfloor \log N \rfloor$	$\lceil \log n \rceil + 6$
Ancestry	Trees	$\lceil \log N \rceil + \lceil \log \lceil \log N \rceil \rceil - 2$	$\log n + 6\log\log n + O(1)$
	Trees(Δ)	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\log n + 6\log\log n + O(1)$
	Trees(D)	$\lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	$\log n + 2\log D + O(1)$
	BinaryTrees	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\log n + 6\log\log n + O(1)$
	Caterpillars	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor + 1$
Sibling (NU)	Trees	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
	Trees(Δ)	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
	Trees(D)	$\lfloor \log N \rfloor - 1$	$\lfloor \log n \rfloor$
	BinaryTrees	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
	Caterpillars	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
Sibling (U)	Trees	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$
	Trees(Δ)	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$
	Trees(D)	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$
	BinaryTrees	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor + 1$
	Caterpillars	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$
NCA	Trees	$\lfloor 1.008 \log N \rfloor - 317$	$\lceil (1 + \log(2+\sqrt{2}))\lfloor \log n \rfloor \rceil$
	Trees(Δ)	$\lfloor 1.008 \log N \rfloor - 317$	$\lceil (1 + \log(2+\sqrt{2}))\lfloor \log n \rfloor \rceil$
	Trees(D)	$\lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	$\lceil (1 + \log(2+\sqrt{2}))\lfloor \log n \rfloor \rceil$
	BinaryTrees	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lceil (1 + \log 3)\lfloor \log n \rfloor \rceil + 1$
	Caterpillars	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil + 1$
Distance	Graphs	$\lfloor \frac{1}{2}N \rfloor - 1$	$\lfloor (\log 3)n \rfloor + \lceil \log n \rceil$
	Trees	$\lfloor \frac{1}{2}\lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$
	Trees(Δ)	$\lfloor \frac{1}{2}\lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$
	Trees(D)	$\lfloor \log N \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$
	BinaryTrees	$\lfloor \frac{1}{2}\lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (\frac{3}{2} + \log 3)\lceil \log n \rceil \rceil + 1$
	Caterpillars	$2\lfloor \log N \rfloor - \lfloor \log \lfloor \log N \rfloor \rfloor - 4$	$2\lfloor \log n \rfloor$

Table 1.2: Some of the consequences of the results in Table 1.1.



Chapter 2

Adjacency

Adjacency labeling schemes for general graphs were studied as early as in the 1960es in a restricted form based on Hamming distances [12, 13]. The more general definition that we use today was introduced by Kannan, Naor and Rudich [27] who described the adjacency labeling scheme from Section 1.2 together with extensions to larger graph families.

We define the *adjacency function* for any graph G and any nodes $u, v \in G$ by

$$\mathrm{adj}(u,v) = \begin{cases} \mathsf{true}, & \text{if } uv \text{ is an edge in } G, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

Note that we define nodes to *not* be adjacent to themselves. An *adjacency labeling scheme for* $\mathcal{G}$ is defined for any subfamily $\mathcal{G} \subseteq \mathsf{Graphs}$ as the special case of the definition in Section 1.4 with $f = \mathrm{adj}$, $k = 2$ and $S = \{\mathsf{true}, \mathsf{false}\}$. Table 2.1 shows some of the lower and upper bounds of the corresponding *adjacency labeling problem*.

It is not uncommon in the literature to require adjacency labeling schemes to produce *unique* labels. In the cases we shall consider, this restriction does not make a difference for the worst-case label sizes, since it will never be the case that it is even possible to assign identical labels to two distinct nodes. In fact, if two nodes can be given the same label, then they must be connected by an edge to *exactly* the same set of nodes; in particular, there cannot be an edge between the two nodes. In a tree, the only nodes with this property are sibling leaf nodes. Although we do not make unique labels a formal requirement for adjacency labeling schemes, all the schemes encountered in this thesis do, in fact, produce unique labels.

Adjacency in trees is tightly linked to *parenthood* (given two nodes, is the first a parent of the second?), since two nodes are adjacent exactly when one is the parent of the other. A parent labeling scheme can easily be transformed into an adjacency labeling scheme by simply extending the decoder algorithm to check not only if u is a parent of v but also if v is a parent of u. Conversely, an adjacency labeling scheme can be transformed into a parent labeling scheme by prefixing each label with two bits used to indicate the depth of the node modulo 3: if u and v are adjacent and the depth of u is one lower than the depth of

Family	Bound on label size	Reference
$\mathsf{Graphs}(N)$	$L \geq \lfloor \frac{1}{2}N \rfloor - 1$	Theorem 2.1, [34, 21]
$\mathsf{Trees}(N, D)$	$L \geq \lfloor \log N \rfloor$	Theorem 2.2
$\mathsf{Paths}(N)$	$L \geq \lfloor \log N \rfloor$	Theorem 2.2
Graphs	$\ell \leq \lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$	Theorem 2.3, [34]
Trees	$\ell \leq \log n + O(\log^* n)$	Theorem 2.4, [10]
Trees	$\ell \leq \log n + 3\log d + 2\log\log d + O(1)$	Theorem 2.5, [17]
$\mathsf{Trees}(N)$	$\ell \leq \log N + 3\log d + O(1)$	Theorem 2.5, [17]
$\mathsf{Trees}(D)$	$\ell \leq \log n + 3\log D + O(1)$	Theorem 2.5, [17]
$\mathsf{Caterpillars}$	$\ell \leq \lceil \log n \rceil + 6$	Theorem 2.6, [11]
$\mathsf{BinaryTrees}$	$\ell \leq \log n + O(1)$	Theorem 2.7, [11]

Table 2.1: Bounds on the optimal worst-case label size for the adjacency labeling problem for different families of graphs. See Section 1.7 for further detail.

Family $\mathcal{G}$	Lower bound on L for $\mathcal{G}(N)$	Upper bound on ℓ for $\mathcal{G}$
Graphs	$\lfloor \frac{1}{2}N \rfloor - 1$	$\lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$
Trees	$\lfloor \log N \rfloor$	$\log n + O(\log^* n)$
$\mathsf{Trees}(\Delta)$	$\lfloor \log N \rfloor$	$\log n + O(\log^* n)$
$\mathsf{Trees}(D)$	$\lfloor \log N \rfloor$	$\log n + 3\log D + O(1)$
$\mathsf{BinaryTrees}$	$\lfloor \log N \rfloor$	$\log n + O(1)$
$\mathsf{Caterpillars}$	$\lfloor \log N \rfloor$	$\lceil \log n \rceil + 6$

Table 2.2: Some of the consequences of the results in Table 2.1.

v modulo 3, then u is the parent of v. Thus, adjacency labeling schemes and parent labeling schemes are equivalent up to a difference of at most two bits in the optimal worst-case label size. For this reason, it suffices to investigate only one of these two types of labeling schemes.

Another interesting feature of adjacency labeling schemes is their connection to *induced-universal graphs*. Given a graph family $\mathcal{G}$, a graph U is $\mathcal{G}$-induced-universal if every graph in $\mathcal{G}$ is an induced subgraph of U. As an example, consider, for some fixed N, the graph U_N with node set $\{(i, j) \mid 0 \leq i, j < N\}$ and an edge between (i, j) and (i', j') exactly when $i = j'$ or $j = i'$ but not both. It is not hard to see that this graph is $\mathsf{Trees}(N)$-induced-universal: the encoder of the adjacency labeling scheme described in Section 1.2 provides the needed map from the node set of an arbitrary tree in $\mathsf{Trees}(N)$ to a subset of the nodes in U_N. The graph U_N has N^2 nodes, which should be compared to the labeling scheme's worst-case label size of $2\lceil \log n \rceil \leq 2\lceil \log N \rceil$. A general connection is

shown in [27]: if there exists an adjacency labeling scheme for $\mathcal{G}$ with label sizes bounded by $k \log N$, then there exists a $\mathcal{G}$-induced-universal graph with at most N^k nodes. We will not prove this, nor will we discuss induced-universal graphs any further in this thesis. It is just worth pointing out the connection as an additional motivation for studying adjacency labeling schemes.

The primary focus of this thesis is on trees, but since adjacency is such a fundamental property of a graph, this chapter includes a discussion of general graphs as well.

2.1 Lower bounds

We begin with a lower bound for adjacency labeling schemes for general graphs: that is, for the family Graphs. It has long been known [34] that the worst-case label size for an optimal adjacency labeling scheme for Graphs is $\Omega(n)$, where n is the number of nodes. We here present a more precise result using a simple proof, which is a special case of an argument in [21, Theorem 3.1].

Theorem 2.1. *For any N, any adjacency labeling scheme for $\mathsf{Graphs}(N)$ has a worst-case label size of at least $\lfloor \frac{1}{2}N \rfloor - 1$.*

Proof. Consider the set of graphs with node set $\{1, \ldots, N\}$. Any graph with this node set is determined uniquely by the set of pairs of nodes that are connected by an edge, so there must be $2^{\binom{N}{2}}$ distinct such graphs (although many of these, of course, are isomorphic). A labeling scheme for $\mathsf{Graphs}(N)$ can, in particular, be applied to this kind of graph, which leads to a map from the set of $2^{\binom{N}{2}}$ graphs to the set of N-tuples $(l_1, \ldots, l_N)$ of labels for the nodes $1, \ldots, N$. This map is injective, because if two graphs are distinct, there must be an edge that occurs in one graph but not the other, and at least one of the two nodes corresponding to this edge must therefore be labeled differently in the two graphs, or else the labeling scheme would not work as intended.

Now, if each label has size at most L, then there can be at most $2^{L+1} - 1$ different labels, and hence there can be at most $(2^{L+1} - 1)^N$ N-tuples of labels. Thus, the existence of the injective map from before means that there is an inequality

$$2^{\binom{N}{2}} \leq (2^{L+1} - 1)^N < 2^{(L+1)N},$$

from which it follows that $\binom{N}{2} \leq (L+1)N$, which implies

$$L \geq \lceil \frac{1}{2}(N-1) \rceil - 1 = \lfloor \frac{1}{2}N \rfloor - 1. \qquad \square$$

For trees, a trivial lower bound can be obtained by specifying a collection of trees whose labels must all be distinct. As noted previously, the only requirement for such trees is that every leaf in them is an only child (meaning that it has no siblings besides itself).

Theorem 2.2. *For any N and any $D \geq 2$, any adjacency labeling scheme for* $\mathsf{Paths}(N)$ *or* $\mathsf{Trees}(N, D)$ *has a worst-case label size of at least* $\lfloor \log N \rfloor$.

Proof. A path with N nodes must have N unique labels and is an object of $\mathsf{Paths}(N)$. Likewise, a tree consisting of one root node to which $\lfloor N/2 \rfloor$ paths of length 2 and, if N is odd, one extra path of length 1 have been attached must also have unique labels and is an object of $\mathsf{Trees}(N, D)$. In both cases, if the worst-case label size is L we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $N \leq 2^{L+1} - 1$ from which it follows that $L \geq \lceil \log(N+1) \rceil - 1 = \lfloor \log N \rfloor$. □

2.2 Upper bounds

As for the lower bounds, we begin by looking at general graphs. Given a graph with n nodes, we can label the nodes using exactly $\lceil \log n \rceil$ bits (see Lemma A.2 in Appendix A). Concatenating each node with its row in the adjacency matrix gives a label of total size $\lceil \log n \rceil + n$. Since the total label length $\lceil \log n \rceil + n$ uniquely determines n, no extra information is needed in order to be able to split up a label into its two subcomponents. Given two such labels, we can determine adjacency by reading the number of the first node and looking up the corresponding entry in the adjacency list for the second node. Thus, there is an adjacency labeling scheme with a worst-case label size of $\lceil \log n \rceil + n$.

This naïve adjacency labeling scheme is not optimal. The fact that we can do better was discovered already in 1964 by Moon [34], who presented both lower and upper bounds on the size of a $\mathsf{Graphs}(N)$-induced-universal graph (see the discussion in the beginning of the chapter). Moon's lower bound corresponds to the lower bound established in Theorem 2.1, and his upper bound corresponds to an upper bound for worst-case label sizes in an adjacency labeling scheme for $\mathsf{Graphs}(N)$ of approximately

$$\begin{cases} \frac{1}{2}(N-1) + \log N - 1, & \text{if } N \text{ is odd,} \\ \frac{1}{2}(N-1) + \log N - 0.915, & \text{if } N \text{ is even} \end{cases}$$

Moon's universal graph can even be extended to one that works for all of Graphs (which implies that it must be infinitely large), meaning that the corresponding labeling scheme can be made to work for Graphs too with the same worst-case label sizes as above (using "n" instead of "N").

We here present a simple labeling scheme for Graphs with a worst-case label size of $\lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$, no matter the parity of n, following the motto that the simplicity is worth the extra bit or two. The scheme is a modification of the naïve labeling scheme from above. In the naïve scheme, each node stores a full row in the adjacency matrix, meaning that the n nodes combined store all n^2 entries of the adjacency matrix. However, since the adjacency matrix is symmetric and

always has 0s in the diagonal, it suffices to store only $\frac{1}{2}n(n-1)$ of the entries, which corresponds to storing $\frac{1}{2}(n-1)$ bits at each node. What remains is to find a way to divide the $\frac{1}{2}n(n-1)$ entries among the n nodes in such a way that, for any pair (i,j), $0 \leq i,j < n$, either the i'th or the j'th node will have stored the (i,j) entry of the adjacency matrix. Our solution is to store only the following entries from the adjacency matrix:

	0	1	2	3	4	5	6	⋯
0			•		•		•	
1	•			•		•		
2		•			•		•	
3	•		•			•		⋯
4		•		•			•	
5	•		•		•			
6		•		•		•		
⋮				⋮				

Theorem 2.3. *There exists an adjacency labeling scheme for* Graphs *whose worst-case label size is at most* $\lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$.

Proof. The special case where $n = 1$ can be handled by giving the single node the empty string as label, and letting this case be the only situation in which a node is given the empty string as label.

For $n > 1$, enumerate each node with the numbers 0 through $n-1$ using exactly $\lceil \log n \rceil$ bits (see Lemma A.2 in Appendix A). The label of the i'th node will be the concatenation of these $\lceil \log n \rceil$ bits and all the (i,j) entries of the adjacency matrix for all j's satisfying

$$\left.\begin{array}{l} j \text{ is odd and } j < i, \text{ or} \\ j \text{ is even and } j > i \end{array}\right\} \text{ if } i \text{ is even}$$

$$\left.\begin{array}{l} j \text{ is odd and } j > i, \text{ or} \\ j \text{ is even and } j < i \end{array}\right\} \text{ if } i \text{ is odd}$$

If n is odd, we append an extra trash bit to the label of all odd numbered nodes. This gives a label of exactly $\lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$ bits for all nodes.

Now, if the numbers i and j of two nodes have the same parity, then the node with the smallest number has the (i,j) entry of the adjacency matrix stored in its label, whereas if the two numbers have different parity, then the node with the largest number has the (i,j) entry of the adjacency matrix stored in its label. Thus, the decoder only needs to decompose the labels into the $\lceil \log n \rceil$ bits giving its number and the $\lfloor \frac{1}{2}n \rfloor$ bits giving the corresponding entries in the adjacency

matrix, after which the adjacency between the two nodes can be determined. Because of the extra trash bit, all labels consist of exactly $\lfloor \frac{1}{2}n \rfloor + \lceil \log n \rceil$ bits, and since this expression increases monotonically with n, the decoder can uniquely determine n from the size of the label and will therefore know how to split up the label: if it sees that n is odd and that the node has an odd number, it will discard the extra trash bit. □

Comparing this result with the lower bound from Theorem 2.1, we see that the adjacency problem for general graphs can be considered closed, since the lower and upper bounds match in the dominating term of $\frac{1}{2}n$. Any attempt to close the insignificant gap of around $\log n$ would primarily be for aesthetic reasons, and, indeed, there have been remarkably few results on the subject since Moon's lower and upper bounds from 1964. Nonetheless, aesthetics is also important and may lead to new insights, so it would be nice to see the gap closed in the future.

We now move on to upper bounds for trees, which seem to have received a lot more attention than the upper bound for general graphs. The to date smallest known upper bound of $\log n + O(\log^* n)$ was given by Alstrup and Rauhe [10]. We shall not go into the details of the rather intricate proof here but present instead a sketch of the proof.

Theorem 2.4. *There exists an adjacency labeling scheme for* Trees *whose worst-case label size is at most* $\log n + O(\log^* n)$.

Proof sketch. The proof begins with the construction of a simple adjacency labeling scheme with labels of size $\log n + 3 \log \log n + O(1)$. The scheme is based on the numbers $\mathrm{dfs}(v)$ given to each node v through a depth-first traversal of the tree, in which children of small size are visited before children of larger size. The labeling scheme stores, for each node v, the numbers $\mathrm{dfs}(v)$, $\mathrm{ldepth}(v)$, $\lfloor \log(\mathrm{dfs}(v) - \mathrm{dfs}(u)) \rfloor$ and $\lfloor \log(\mathrm{dfs}(w) - \mathrm{dfs}(v)) \rfloor$, where u is the parent of v, w is a child of v with maximum value of $\mathrm{dfs}(w)$ (and hence of maximum size), and $\mathrm{ldepth}(v)$ is the *light depth* of v, which will be explained in Section 5.2.1. This information is sufficient to determine if two nodes are adjacent or not.

The labeling scheme is then extended as follows. In addition to the existing label, all internal nodes also encode the logarithm of (a representative of) the number of leaf children of the node. This increases the label size for internal nodes to $\log n + 4 \log \log n + O(1)$. All leaves, on the other hand, can now have their label reduced to only include their depth-first number of size $\log n + O(1)$.

The final step of the label construction is somewhat involved and will only be superficially described. It is a recursively iterated combination of two labeling schemes, A and B, where A uses many bits for internal nodes and few for leaf nodes, whereas B does the opposite. The construction starts with A and B equal to the previously described labeling schemes, and they are combined using a so-called *cluster partitioning* of the tree. The combination procedure defines a labeling scheme with a better worst-case label size than that of B. Recursively

reapplying the procedure with B replaced by the new labeling scheme therefore leads to a series of labeling schemes with smaller and smaller worst-case label size until an equilibrium of $\log n + O(\log^* n)$ is reached. □

For a tree with very small depth, a label size of $\log n + 3\log d + O(1)$ may be smaller than one of $\log n + O(\log^* n)$. The theorem below from [17] shows that there is an adjacency labeling scheme with labels of at most this size. The labeling scheme is obtained by a small modification of an ancestry labeling scheme, which we shall return to in Theorem 3.5.

Theorem 2.5. *For all N, there exists an adjacency labeling scheme for* $\mathsf{Trees}(N)$ *whose worst-case label size is at most* $\log N + 3\log d + O(1)$. *Further, for all D, there exists an adjacency labeling scheme for* $\mathsf{Trees}(D)$ *whose worst-case label size is at most* $\log n + 3\log D + O(1)$. *Finally, there exists an adjacency labeling scheme for* Trees *whose worst-case label size is at most* $\log n + 3\log d + 2\log\log d + O(1)$.

Proof sketch. Theorem 3.5 presents ancestry labeling schemes for the families $\mathsf{Trees}(N)$, $\mathsf{Trees}(D)$ and Trees with worst-case label sizes of $\log N + 2\log d + O(1)$, $\log n + 2\log D + O(1)$ and $\log n + 2\log d + 2\log\log d + O(1)$, respectively. Adjacency of two nodes can be determined from from ancestry and depth: two nodes are adjacent if and only if one is an ancestor of the other and their depths differ by exactly 1. The aforementioned ancestry labeling scheme can therefore be converted into adjacency labeling schemes by adding the depth of each node as a suffix to its label, which can be done using $\lceil \log d \rceil \leq \lceil \log D \rceil$ bits, and modifying the decoder so that it can retrieve the depth and the original label from the concatenation. □

Even though the general adjacency problem for trees is still open, the problem has been closed in the important special cases of binary trees and caterpillars. This was proven by Bonichon, Gavoille and Labourel [11] using a technique known as "traversal and jumping". The technique is so called since internal nodes are associated with intervals that are constructed through a traversal of the tree and with sufficient jumps from one interval to the next to allow an efficient encoding. We shall not go into the details of the proofs here, but just sketch the general idea.

Theorem 2.6. *There exists an adjacency labeling scheme for* $\mathsf{Caterpillars}$ *whose worst-case label size is at most* $\lceil \log n \rceil + 6$.

Proof sketch. Denote the i'th node on the main path by x_i and its j'th child node (not on the main path) by $y_{i,j}$. The labeling scheme defines labels $l(x_i)$ and $l(y_{i,j})$ for these nodes and, for each i, numbers p_i such that the sets $I_i = \{l(x_i) + 1, \ldots, l(x_i) + p_i\}$ are pairwise disjoint; such that $l(y_{i,j}) \in I_i$ for all i, j; and such that $l(x_i) + p_i < l(x_j) \leq l(x_i) + p_i + p_{i+1}$ holds if and only if $j = i+1$. The numbers p_i are chosen in the form $p_i = 2^{t_i+3}$. Using an extra bit to distinguish

nodes on the main path from other nodes, it is now easy to determine adjacency between any two nodes from their labels and the numbers t_i. It is not hard to choose labels of size $\log n + O(1)$ with the above properties, but the problem is how to get access to the numbers t_i. The solution is to choose $l(x_i)$ so that the *number itself* contains information about t_i and t_{i+1}. This is done using an efficient encoding of (t_i, t_{i+1}) and a small lemma stating that, for a binary string w and an integer z, there exists an integer x with $z \leq x < z + 2^{|w|}$ such that w is a suffix of the binary representation of x. □

Theorem 2.7. *There exists an adjacency labeling scheme for* BinaryTrees *whose worst-case label size is at most* $\log n + O(1)$.

Proof sketch. The idea is similar to that of Theorem 2.6 and will only be superficially described. First, we assume that every internal node has two children: this can be obtained by at most doubling the number of nodes, which just adds a single bit to the final label size. For an internal node, denote by v^- and v^+ the left and right children of v, respectively. The labels $l(v)$ are constructed so that adjacency corresponds to containment in intervals $I_v^- = \{l(v) + s(v), \ldots, l(v) + s(v)+s(v^-)-1\}$ and $I_v^+ = \{l(v)+s(v)+q(v^-), \ldots, l(v)+s(v)+q(v^-)+s(v^+)-1\}$, where $s(v)$ and $q(v)$ are numbers associated with each node v. More specifically, if $l(v) < l(w)$, then v and w are adjacent if and only if v is inner and either $l(w) \in I_v^-$ (in this case $w = v^-$) or $l(w) \in I_v^+$ (in this case $w = v^+$). The labels $l(v)$ are now chosen, using the same trick as in Theorem 2.6, such that the above is satisfied and such that the numbers $s(v^-)$, $s(v^+)$ and $q(v^-)$ can be extracted directly from $l(v)$ itself. □

Chapter 3

Ancestry

The ancestry labeling problem was among the problems discussed by Kannan, Naor and Rudich [27] when the modern concept of labeling schemes was introduced. They present the following simple $2\lceil \log n \rceil$ ancestry labeling scheme. Given a tree with n nodes, first number each node v with the number $\mathrm{dfs}(v)$ obtained through a depth-first traversal of the tree. Now label each node v with the integer interval $I(v) = \{\mathrm{dfs}(v), \ldots, \mathrm{dfs}(w)\}$, where w is the descendant of v with the maximum value of $\mathrm{dfs}(w)$. The interval can be encoded as the pair of endpoints using $2\lceil \log n \rceil$ bits, and ancestry queries can now be answered with a simple interval containment test: u is the ancestor of v if and only if $I(v) \subseteq I(u)$.

This simple labeling scheme initiated an extensive research [2, 28, 29, 9, 1, 17], culminating in a matching lower bound [5] (Theorem 3.1) and upper bound [18] (Theorem 3.4), yielding that an ancestry labeling scheme must have a worst-case label size of $\log n + \Theta(\log \log n)$. As one of the few labeling problems, the ancestry problem for trees is therefore generally considered closed.

We define the *ancestry function* for any tree T and any nodes $u, v \in T$ by

$$\mathrm{anc}(u, v) = \begin{cases} \mathsf{true}, & \text{if } u \text{ is an ancestor of } v, \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

Note that a node is always assumed to be its own ancestor. An *ancestry labeling scheme for* $\mathcal{T}$ is defined for any subfamily $\mathcal{T} \subseteq \mathsf{Trees}$ as the special case of the definition in Section 1.4 with $f = \mathrm{anc}$, $k = 2$ and $S = \{\mathsf{true}, \mathsf{false}\}$. Table 3.1 shows some of the lower and upper bounds of the corresponding *ancestry labeling problem*.

Since we have assumed nodes to be their own ancestors, an adjacency labeling scheme must produce unique labels: the decoder will always return true when given two identical labels, so if two nodes have the same label, they must be each other's ancestor, and hence they must be identical. If we had not defined nodes to be their own ancestor, it would have been possible for distinct nodes to have identical labels, although only when the nodes have *exactly* the same set of ancestors and descendants, which happens only for sibling leaf nodes. In this case, we could still make it a formal requirement for the labeling scheme to produce unique labels, and in this case, the concept is exactly equivalent to our

Family	Bound on label size	Reference
$\mathsf{Trees}(N, D)$	$L \geq \lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	Theorem 3.1, [5]
$\mathsf{BinaryTrees}(N)$	$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	Theorem 3.2, [5]
$\mathsf{Paths}(N)$	$L \geq \lfloor \log N \rfloor$	Theorem 3.3
Trees	$\ell \leq \log n + 6 \log \log n + O(1)$	Theorem 3.4, [18]
$\mathsf{Trees}(N)$	$\ell \leq \log N + 4 \log \log N + O(1)$	Theorem 3.4, [18]
Trees	$\ell \leq \log n + 2 \log d + 2 \log \log d + O(1)$	Theorem 3.5, [18]
$\mathsf{Trees}(N)$	$\ell \leq \log N + 2 \log d + O(1)$	Theorem 3.5, [18]
$\mathsf{Trees}(D)$	$\ell \leq \log n + 2 \log D + O(1)$	Theorem 3.5, [18]
$\mathsf{Caterpillars}$	$\ell \leq \lfloor \log n \rfloor + 1$	Theorem 3.6

Table 3.1: Bounds on the optimal worst-case label size for the ancestry labeling problem for different families of trees. See Section 1.7 for further detail.

Family $\mathcal{G}$	Lower bound on L for $\mathcal{G}(N)$	Upper bound on ℓ for $\mathcal{G}$
Trees	$\lceil \log N \rceil + \lceil \log \lceil \log N \rceil \rceil - 2$	$\log n + 6 \log \log n + O(1)$
$\mathsf{Trees}(\Delta)$	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\log n + 6 \log \log n + O(1)$
$\mathsf{Trees}(D)$	$\lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	$\log n + 2 \log D + O(1)$
$\mathsf{BinaryTrees}$	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\log n + 6 \log \log n + O(1)$
$\mathsf{Caterpillars}$	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor + 1$

Table 3.2: Some of the consequences of the results in Table 3.1.

version: the only difference lies in whether the decoder returns false or true when given two identical labels. But even if we do not require unique labels, the two concepts turn out to be equivalent in the sense that the worst-case label size of an optimal ancestry labeling scheme is the same in both cases. This follows by noting that the trees used in Theorem 3.1 for the lower bound are forced to have unique labels even if it were not required, whereas the labeling scheme used in Theorem 3.4 for the upper bound does, in fact, produce unique labels in any case.

3.1 Lower bounds

The following two lower bound theorems use the "boxes and groups" technique by Alstrup, Bille and Rauhe [5] to obtain lower bounds that extend their original result in [5]. The boxes and groups technique is described in Lemma A.9 in Appendix A.

Theorem 3.1. *For any N, D with $N \geq D+1 \geq 3$, any ancestry labeling scheme for* $\mathsf{Trees}(N, D)$ *has a worst-case label size of at least* $\lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$.[1]

Proof. Let $m = 2^{\lfloor \log(N-1) \rfloor} = 2^{\lceil \log N \rceil - 1}$ be $N-1$ rounded down to the nearest power of 2, and set $k = \lfloor \log D \rfloor$. Note that $k \leq \log m$. Construct for $i = 1, \ldots, k$ the tree T_i as a root node to which $m/2^i$ paths of length 2^i have been attached. Thus, T_i has $m + 1 \leq N$ nodes, whereof m belong to disjoint paths. Further, T_i has depth $2^i \leq 2^k \leq D$, and hence T_i belongs to $\mathsf{Trees}(N, D)$.

As previously mentioned, each node in a tree must be uniquely labeled by an ancestry labeling scheme. Further, if two nodes in T_i lie on distinct paths, then their labels cannot be used on the same path in T_j for any $j \neq i$, because nodes on the same path have an ancestry relation whereas nodes on different paths do not. We can therefore apply Lemma A.9, using the m nodes of the paths of each tree as a "box" and each path as a "group", and it follows that we need at least $\frac{1}{2}m(k+1) = \frac{1}{2}m(\lfloor \log D \rfloor + 1)$ labels. If the worst-case label size is L we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $\frac{1}{2}m(\lfloor \log D \rfloor + 1) \leq 2^{L+1} - 1$ from which it follows that $L \geq \lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$. □

Theorem 3.2. *For any $N \geq 2$, any ancestry labeling scheme for* $\mathsf{BinaryTrees}(N)$ *has a worst-case label size of at least* $\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$.

Proof. Let $n = 2^{\lfloor \log N \rfloor}$ be N rounded down to the nearest power of 2, and set $m = n/2$ and $k = \log m$. Construct for $i = 1, \ldots, k$ the tree T_i as the perfect binary tree with depth $\log m - i$ and $m/2^{i-1} - 1$ nodes, where each of the $m/2^i$ leaves has been extended to a path of length 2^i. Thus, T_i has $m/2^{i-1} - 1 + m < n \leq N$ nodes, whereof m belong to the path extensions. Further, every node in T_i has at most 2 children and hence belongs to $\mathsf{BinaryTrees}(N)$.

Each node in a tree must be uniquely labeled by an ancestry labeling scheme, and if two nodes in T_i lie on distinct path extensions, then their labels cannot be used on the same path extension in T_j for some $j \neq i$, because nodes on the same path extension have an ancestry relation whereas nodes on different path extensions do not. We can therefore apply Lemma A.9, using the m nodes of the path extensions of each tree as a "box" and each path extension as a "group", and it follows that we need at least $\frac{1}{2}m(k+1) = \frac{1}{4}n \log n$ labels. If the worst-case label size is L we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $\frac{1}{4}n \log n < 2^{L+1} - 1$ from which it follows that $L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$. □

Since none of the trees in Theorems 3.1 and 3.2 are caterpillars, we cannot use them to obtain a lower bound for $\mathsf{Caterpillars}$ or $\mathsf{Caterpillars}(N)$. Instead, we simply remark that the uniqueness of labels does give us a trivial lower bound of $\lfloor \log n \rfloor$ in a tree with n nodes. We state this observation as a general theorem for paths (which are a special case of caterpillars) in the theorem below. As it turns out, the trivial lower bound for caterpillars is optimal; see theorem 3.6.

[1] Observe that the assumption $N \geq D+1$ is natural, since any tree with n nodes and depth d satisfies $n \geq d+1$.

Theorem 3.3. *For any N, any ancestry labeling scheme for* $\mathsf{Paths}(N)$ *has a worst-case label size of at least* $\lfloor \log N \rfloor$.

3.2 Upper bounds

The simple ancestry labeling scheme from [27] described in the beginning of this chapter translates ancestor queries into inclusion of integer intervals. This idea is also used by Fraigniaud and Korman [18], who were the first to describe an optimal (up to the dominant term) ancestry labeling scheme. We shall not prove the theorem in detail but simply outline the idea.

Theorem 3.4. *For any N, there exists an ancestry labeling scheme for* $\mathsf{Trees}(N)$ *whose worst-case label size is at most* $\log N + 4 \log \log N + O(1)$. *Further, there exists an ancestry labeling scheme for* Trees *whose worst-case label size is at most* $\log n + 6 \log \log N + O(1)$.

Proof sketch. We begin with the labeling scheme for $\mathsf{Trees}(N)$. The scheme uses Harel and Tarjan's *heavy-light decomposition* [25], which will be described in Section 5.2.1. Let $\text{apex}(v)$ denote the *apex* of v from the heavy-light decomposition, and let $\text{lqa}(v)$ denote the set of *local quasi-ancestors* of v, which are all the nodes on the path between, but not including, $\text{parent}(v)$ and $\text{apex}(\text{parent}(v))$ as well as all their light children, but excluding all nodes that have a depth-first search number higher than that of v. Further, for integer intervals I and J, let $I \prec J$ mean that $x < y$ for all $x \in I$ and $y \in J$.

Each node v is associated with an integer interval $I(v)$ satisfying that, when v is not the root, $I(v) \subseteq I(\text{apex}(v)) \cap I(\text{apex}(\text{parent}(v))$ and $I(w) \prec I(v)$ for all $w \in \text{lqa}(v)$. This property implies that u is a proper ancestor of v if and only if $I(v) \subset I(\text{apex}(u))$ and either $I(u) \prec I(v)$ or $I(u) = I(\text{apex}(u))$. It follows that it suffices to label each node v with an encoding of the intervals $I(v)$ and $I(\text{apex}(v))$.

The intervals are constructed (or rather: chosen) recursively from the heavy-light decomposition. The construction starts with the root r being associated with the interval $I(r)$. The nodes on the same heavy path as r are then associated with disjoint subintervals of $I(r)$, ordered by $\prec$ according to the order on the heavy path. For each node v on the heavy path of r, the construction then continues recursively for the subtrees consisting of v and all descendants of the light children of v. The intervals are chosen in the form $\{2^i a, \ldots, 2^i(a+b)\}$, where $1 \leq i \leq \log N$, $1 \leq a \leq 2^{2-i} N \log N$ and $1 \leq b \leq 4i$, meaning that an interval can be encoded by encoding the numbers i, a and b. These three numbers can be encoded with $\log N + \log \log N + O(1)$ bits. To encode both $I(v)$ and $I(\text{apex}(v))$, it suffices to note that the numbers i', a', b' defining $I(\text{apex}(v))$ can be derived from the numbers i, a, b defining $I(v)$ using only an additional $3 \log \log N + O(1)$ bits, giving a total of $\log N + 4 \log \log N + O(1)$ bits.

The above labeling scheme is constructed for $\mathsf{Trees}(N)$ and assumes knowledge of $\log N$ in order to work. Another option is to encode the value $\log n$, where n is the number of nodes in an individual tree, into each label in order to construct a labeling scheme for Trees that is independent of any knowledge of N. This can be done with an additional $2\log\log n + O(1)$ bits (for example by prefixing the label with $\mathtt{0}^{|x|-1}\mathtt{1} \cdot x$, where x is the binary representation of $\log n$), resulting in a labeling scheme with $\log n + 6\log\log n + O(1)$ bits. Note that, although this label size is larger than $\log N + 4\log\log N + O(1)$ when $n = N$, it will be substantially smaller for trees with $n \ll N$ nodes. □

Comparing with Theorems 3.1 and 3.2 we see that the optimal worst-case label size for Trees, $\mathsf{Trees}(N)$, $\mathsf{Trees}(\Delta)$, $\mathsf{BinaryTrees}$ and $\mathsf{BinaryTrees}(N)$ in any case is $\log N + \Theta(\log\log N)$, which means that the ancestry problem is generally considered closed for these families. There is, of course, still a gap between the lower and upper bounds, but the gap is asymptotically insignificant since the $\log N$ term dominates the expression no matter what. Nonetheless it would still be interesting to see if the gap could be *completely* closed, at least up to the constant term.

By considering also the depth of a tree as a parameter that can be part of the formula for label sizes, [18] also presents a labeling scheme with worst-case label sizes expressed in terms of both n and d. The result is presented in the theorem below. Note that the depth has to be significantly smaller than the number of nodes for this labeling scheme to have smaller worst-case label size than the scheme in Theorem 3.4.

Theorem 3.5. *For any N, there exists an ancestry labeling scheme for $\mathsf{Trees}(N)$ whose worst-case label size is at most $\log N + 2\log d + O(1)$. Further, for any D, there exists an ancestry labeling scheme for $\mathsf{Trees}(D)$ whose worst-case label size is at most $\log n + 2\log D + O(1)$. Finally, there exists a labeling scheme for Trees whose worst-case label size is at most $\log n + 2\log d + 2\log\log d + O(1)$.*

Proof sketch. The strategy is similar to that of Theorem 3.4 and will only be superficially described. We begin by constructing a labeling scheme for $\mathsf{Trees}(N, D)$. As in Theorem 3.4 the encoder algorithm uses Harel and Tarjan's heavy-light decomposition [25], which will be described in Section 5.2.1, but this time in a version where a node is only considered heavy if its size accounts for more than half of the size of its apex (the authors of [18] denote this *spine decomposition*). Also as in Theorem 3.4, the decoder recursively assigns an interval $I(v)$ to each node v such that ancestry can be determined from a simple interval containment test; in fact, the test is simpler now in that u is an ancestor of v if and only if $I(v) \subseteq I(u)$. The construction of the intervals and their encoding will not be described here, but they are encoded with a total of $\log n + 2\log d + O(1)$ bits, and decoding them requires knowledge of both $\log n$ and $\log d$.

Since the decoder always knows the label size $\log n + 2\log d + O(1)$, it can derive $\log d$ from n or, conversely, $\log n$ from d, meaning that it never needs prior knowledge of both n and d. To obtain a labeling scheme for $\mathsf{Trees}(N)$, we can therefore modify the algorithm to replace n by N in the constructions, after which $\log d$ can be derived from the label size $\log N + 2\log d + O(1)$. Likewise, to obtain a labeling scheme for $\mathsf{Trees}(D)$, we replace d by D in the constructions, after which $\log n$ can be derived from the label size $\log n + 2\log D + O(1)$.

To obtain a labeling scheme for Trees that does not rely on any prior knowledge of N or D, we can encode $\log d$ directly into each label using an additional $2\log\log d + O(1)$ bits (for example by prefixing each label with $\mathsf{0}^{|y|-1}\mathsf{1} \cdot y$, where y is the binary representation of $\log d$), resulting in a labeling scheme for Trees with labels of size $\log n + 2\log d + 2\log\log d + O(1)$. □

Theorem 3.3 presented a trivial lower bound of $\lfloor \log N \rfloor$ for $\mathsf{Paths}(N)$, which also holds for $\mathsf{Caterpillars}(N)$. The matching upper bound is presented in the theorem below:

Theorem 3.6. *There exists an ancestry labeling scheme for* $\mathsf{Caterpillars}$ *whose worst-case label size is at most* $\lfloor \log n \rfloor + 1$.

Proof. We enumerate the nodes of a caterpillar with n nodes as follows. Start at the root and number all leaf children (not on the main path) of the root before moving on to the next node on the main path. Then do the same recursively for the remaining nodes. A labeling of size $\lfloor \log n \rfloor + 1$ can now be created by, for each node, using a single bit to determine whether the node is on the main path and a binary string of length at most $\lfloor \log n \rfloor$ to encode the node's number (see Lemma A.1 in Appendix A). Given a label, the decoder will then be able to determine the corresponding node's number as well as whether it is on the main path or not, and this information suffices to determine ancestry: if the node u has a strictly lower number than the node v, then u is an ancestor of v exactly when u is on the main path. □

Caterpillars are one of the few cases where it would have made a difference if we had defined nodes to not be their own ancestors and had allowed labels to be non-unique: for if this were the case, we could use the same number for all siblings not on the main path, which would reduce the label size to $\lfloor \log m \rfloor + 1$, where m is the length of the main path.

Chapter 4

Sibling

The sibling labeling problem is one of the least discussed labeling problems in the literature. The main reference is [5] which discusses both lower and upper bounds for the sibling problem. The reason for this lack of literature is, most likely, that the problem is not so hard, even if we require labels to be unique. This chapter shows that the sibling labeling problem has been closed (up to a tiny additive constant) for all the families of trees under consideration in this thesis.

We define the *sibling function* for any tree T and any nodes $u, v \in T$ by

$$\mathrm{sib}(u,v) = \begin{cases} \mathsf{true}, & \text{if } u \text{ and } v \text{ are siblings,} \\ \mathsf{false}, & \text{otherwise.} \end{cases}$$

Note that a node is always assumed to be its own sibling. A *sibling labeling scheme for* $\mathcal{T}$ is defined for any subfamily $\mathcal{T} \subseteq \mathsf{Trees}$ as the special case of the definition in Section 1.4 with $f = \mathrm{sib}$, $k = 2$ and $S = \{\mathsf{true}, \mathsf{false}\}$.

Since we have assumed nodes to be their own sibling, it is possible for a sibling labeling scheme to produce non-unique labels by giving the same label to each collection of siblings. This property makes it very easy to define optimal sibling labeling schemes, meaning that the sibling labeling problem is almost trivial. However, if we force labels to be unique, the problem becomes harder, and hence more interesting, and also more applicable to situations where labels are required to uniquely identify the nodes of a tree. This chapter therefore discusses both the *unique* and the *non-unique* variant of sibling labeling schemes and sibling labeling problems. Table 4.1 shows some of the lower and upper bounds for the two problems.

4.1 Lower bounds

We begin with the simple, non-unique case where, even though we do not require labels to be unique, can easily find examples where the majority of labels have to be: trees in which most nodes is an only child.

Theorem 4.1. *For any N, any non-unique sibling labeling scheme for* $\mathsf{Paths}(N)$ *has a worst-case label size of at least* $\lfloor \log N \rfloor$.

Family	Variant	Bound on label size	Reference
$\mathsf{Paths}(N)$	NU	$L \geq \lfloor \log N \rfloor$	Theorem 4.1
$\mathsf{Trees}(N, D)$	NU	$L \geq \lfloor \log N \rfloor - 1$	Theorem 4.2
$\mathsf{Caterpillars}(N, \Delta)$	U	$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor - 2$	Theorem 4.3, [5]
$\mathsf{Trees}(D)$	U	$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	Theorem 4.4, [5]
$\mathsf{Paths}(N)$	U	$L \geq \lfloor \log N \rfloor$	Theorem 4.5
Trees	NU	$\ell \leq \lfloor \log n \rfloor$	Theorem 4.6, [5]
$\mathsf{Trees}(\Delta)$	U	$\ell \leq \lfloor \log n \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$	Theorem 4.7, [5]
$\mathsf{BinaryTrees}$	U	$\ell \leq \lfloor \log n \rfloor + 1$	Theorem 4.7
$\mathsf{Trees}(N)$	U	$\ell \leq \lfloor \log N \rfloor + \lfloor \log \lfloor \log \delta \rfloor \rfloor + 1$	Theorem 4.7, [5]
Trees	U	$\ell \leq \lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$	Theorem 4.7, [5]
Trees	U	$\ell \leq \lfloor \log n \rfloor + 2\lceil \log \lfloor \log \delta \rfloor \rceil + 1$	Theorem 4.7, [5]

Table 4.1: Bounds on the optimal worst-case label size for the non-unique (NU) and unique (U) sibling labeling problem for different families of trees. See Section 1.7 for further detail.

	Family $\mathcal{G}$	Lower bound on L for $\mathcal{G}(N)$	Upper bound on ℓ for $\mathcal{G}$
Non-unique	Trees	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
	$\mathsf{Trees}(\Delta)$	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
	$\mathsf{Trees}(D)$	$\lfloor \log N \rfloor - 1$	$\lfloor \log n \rfloor$
	$\mathsf{BinaryTrees}$	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
	$\mathsf{Caterpillars}$	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor$
Unique	Trees	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$
	$\mathsf{Trees}(\Delta)$	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$
	$\mathsf{Trees}(D)$	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$
	$\mathsf{BinaryTrees}$	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor + 1$
	$\mathsf{Caterpillars}$	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$

Table 4.2: Some of the consequences of the results in Table 4.1.

Proof. A path with N nodes must have unique labels and is an object of $\mathsf{Paths}(N)$. If the worst-case label size is L we can create $2^{L+1}-1$ distinct labels, and we must therefore have $N \leq 2^{L+1}-1$ from which it follows that $L \geq \lfloor \log N \rfloor$. □

Theorem 4.2. *For any N, D with $N \geq D+1 \geq 3$, any non-unique sibling labeling scheme for $\mathsf{Trees}(N, D)$ has a worst-case label size of at least $\lfloor \log N \rfloor - 1$.*

Proof. A tree consisting of one root node to which $\lfloor (N-1)/2 \rfloor$ paths of length 2 have been attached has depth $2 \leq D$ and at most N nodes, and is therefore an object of $\mathsf{Trees}(N, D)$. Further, this tree must have unique labels for all nodes except the group of children of the root which can share a single label among them. In total, this requires $\lfloor (N-1)/2 \rfloor + 2 \geq N/2$ unique labels. If the worst-case label size is L we can create $2^{L+1}-1$ distinct labels, and we must therefore have $N/2 \leq 2^{L+1}-1$ from which it follows that $L \geq \lfloor \log N \rfloor - 1$. □

In the unique case, we can establish higher lower bounds using the "boxes and groups" technique by Alstrup, Bille and Rauhe [5]; see Lemma A.9 in Appendix A. Our results extend the original result in [5].

Theorem 4.3. *For any N, Δ with $N \geq \Delta + 1 \geq 4$, any unique sibling labeling scheme for $\mathsf{Caterpillars}(N, \Delta)$ has a worst-case label size of at least $\lfloor \log N \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor - 2$.*[1]

Proof. Let $n = 2^{\lfloor \log N \rfloor}$ be N rounded down to the nearest power of 2 and set $m = n/2$ and $k = \lfloor \log \Delta \rfloor - 1$. Note that $k \leq \log m$ and that $k \leq \log(\Delta - 2)$. Construct for $i = 1, \ldots, k$ the tree T_i as the caterpillar whose main path has length $m/2^i$ and where each node on the main path has 2^i children not on the path. Thus, T_i has $m + m/2^i \leq n \leq N$ nodes, whereof m are leaves not belonging to the main path. Further, each T_i has maximum degree $2 + 2^i \leq 2 + 2^k \leq \Delta$ and hence belongs to $\mathsf{Caterpillars}(N, \Delta)$.

If two nodes in T_i are not siblings, then their labels clearly cannot be used for a pair of siblings in T_j for any $j \neq i$. We can therefore apply Lemma A.9, using the m nodes not belonging to the main path of each tree as a "box" and each collection of siblings not on the main path as a "group", and it follows that we need at least $\frac{1}{2}m(k+1) = \frac{1}{4}n\lfloor \log \Delta \rfloor$ labels. If the worst-case label size is L we can create $2^{L+1}-1$ distinct labels, and we must therefore have $\frac{1}{4}n\lfloor \log \Delta \rfloor \leq 2^{L+1}-1$ from which it follows that $L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor - 2$. □

Theorem 4.4. *For any $D \geq 2$, any unique sibling labeling scheme for $\mathsf{Trees}(N, D)$ has a worst-case label size of at least $\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$.*

Proof. Let $n = 2^{\lfloor \log N - 1 \rfloor} = 2^{\lceil \log N \rceil - 1}$ be $N-1$ rounded down to the nearest power of 2 and set $m = n/2$ and $k = \log m$. Construct for $i = 1, \ldots, k$ the tree

[1] Observe that the assumption $N \geq \Delta + 1$ is natural, since any tree with n nodes and maximum degree δ satisfies $n \geq \delta + 1$.

T_i as a root node with $m/2^i$ children all of which have 2^i children. Thus, T_i has $m + m/2^i + 1 \leq n + 1 \leq N$ nodes, whereof m are leaves. Further, each T_i has depth $2 \leq D$ and hence belongs to $\mathsf{Trees}(N, D)$.

If two nodes in T_i are not siblings, then their labels clearly cannot be used for a pair of siblings in T_j for any $j \neq i$. We can therefore apply Lemma A.9, using the m child nodes of each tree as a "box" and each collection of siblings as a "group", and it follows that we need at least $\frac{1}{2}m(k+1) = \frac{1}{4}n \log n$ labels. If the worst-case label size is L we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $\frac{1}{4}n \log n \leq 2^{L+1} - 1$ from which it follows that $L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$. □

Since the trees in Theorems 4.3 and 4.4 are not binary, we cannot use them to obtain a lower bound for $\mathsf{BinaryTrees}$. Instead, we simply remark that the uniqueness of labels, as in Theorem 4.1, does provide us with a lower bound of $\lfloor \log n \rfloor$ in a tree with n nodes:

Theorem 4.5. *For any N, any unique sibling labeling scheme for $\mathsf{Paths}(N)$ has a worst-case label size of at least $\lfloor \log N \rfloor$.*

4.2 Upper bounds

As for the lower bounds, we begin with the simple, non-unique case.

Theorem 4.6. *There exists a non-unique sibling labeling scheme for Trees whose worst-case label size is at most $\lfloor \log n \rfloor$.*

Proof. A graph with n nodes has at most n groups of siblings, so we can assign a unique binary string to each of these using at most $\lfloor \log n \rfloor$ bits (see Lemma A.1 in Appendix A). Using the assigned string for a group of siblings as the label for every node within the group, the decoder can determine if two nodes are siblings by simply comparing their labels. □

In the unique case, we present a whole collection of labeling schemes in a single theorem, since all the schemes are variations of the same theme.

Theorem 4.7. *For all $\Delta \geq 2$, there exists a unique sibling labeling scheme for $\mathsf{Trees}(\Delta)$, whose worst-case label size is at most $\lfloor \log n \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$. In particular, there exists a unique sibling labeling scheme for $\mathsf{BinaryTrees}$ whose worst-case label size is at most $\lfloor \log n \rfloor + 1$. Further, for all N, there exists a unique sibling labeling scheme for $\mathsf{Trees}(N)$ whose worst-case label size is at most $\lfloor \log N \rfloor + \lfloor \log \lfloor \log \delta \rfloor \rfloor + 1$. Finally, there exists two unique sibling labeling schemes for Trees whose worst-case label sizes are at most $\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$ and $\lfloor \log n \rfloor + 2\lceil \log \lfloor \log \delta \rfloor \rceil + 1$, respectively.*

Proof. Given a tree with n nodes and maximum degree δ, associate with each sibling group a weight w equal to the number of siblings in the group. Note

that the sum of all these weight is n. Using Lemma A.3 from Appendix A, we can create a unique label l_1 for each of these groups of siblings using at most $\lfloor \log n - \log w \rfloor$ bits for a group of w siblings. Next, using Lemma A.1 from Appendix A for a group of w siblings, we can create a unique label l_2 for each individual sibling using $\lfloor \log w \rfloor \leq \lfloor \log \delta \rfloor$ bits. The total length of the concatenation $l_1 \cdot l_2$ is now at most $\lfloor \log n - \log w \rfloor + \lfloor \log w \rfloor \leq \lfloor \log n \rfloor$, and if the decoder is able to retrieve the two sub-labels from the concatenation, it can determine if two nodes are siblings: this happens exactly when their first sub-labels are identical. It remains to provide the decoder with a way to determine how to retrieve the two sub-labels. How we do this depends on the graph family.

In the case of $\mathsf{Trees}(\Delta)$, since $|l_2| \leq \lfloor \log w \rfloor \leq \lfloor \log \Delta \rfloor$, we can encode the length $|l_2|$ as binary number l_0 of length *exactly* $\lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$. We now create the final label $l = l_0 \cdot l_1 \cdot l_2$ of length at most $\lfloor \log n \rfloor + \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$. Since the decoder already knows Δ, it knows $|l_0| = \lfloor \log \lfloor \log \Delta \rfloor \rfloor + 1$ and can therefore retrieve l_0, from which it can read $|l_2|$ and thereby retrieve l_1 and l_2.

The case of $\mathsf{BinaryTrees}$ follows immediately from the preceding by setting $\Delta = 3$.

In the case of $\mathsf{Trees}(N)$, we start by creating the string $l_0 = \mathtt{0}^i\mathtt{1}$, where $i = \lfloor \log N \rfloor - |l_1| - |l_2| \geq 0$. The concatenation $l_0 \cdot l_1 \cdot l_2$ now has length exactly equal to $\lfloor \log N \rfloor + 1$, and the bits belonging to $l_1 \cdot l_2$ are those coming after the first $\mathtt{1}$. Next, we use Lemma A.1 to encode the length $|l_2|$ as a string l_{-1} with $\lfloor \log |l_2| \rfloor \leq \lfloor \log \lfloor \log \delta \rfloor \rfloor$ bits. We now create the final label $l = l_{-1} \cdot l_0 \cdot l_1 \cdot l_2$ of length at most $\lfloor \log N \rfloor + \lfloor \log \lfloor \log \delta \rfloor \rfloor + 1$. Since the decoder knows N, it knows $|l_0 \cdot l_1 \cdot l_2|$ and can therefore retrieve l_{-1} and $l_0 \cdot l_1 \cdot l_2$. From the former it can retrieve $|l_2|$, and from the latter it can retrieve $l_1 \cdot l_2$, and thereby also l_1 and l_2.

In the case of Trees, we create two labeling schemes. The first is a variant of the one for $\mathsf{Trees}(N)$. We first create the string $l_0 = \mathtt{0}^i\mathtt{1}$, where $i = \lfloor \log n \rfloor - |l_1| - |l_2| \geq 0$. The concatenation $l_0 \cdot l_1 \cdot l_2$ now has length exactly equal to $\lfloor \log n \rfloor + 1$, and the bits belonging to $l_1 \cdot l_2$ are those coming after the first $\mathtt{1}$. Next, we use Lemma A.1 to encode the length $|l_2|$ as a string l_{-1} with $\lfloor \log |l_2| \rfloor \leq \lfloor \log \lfloor \log \delta \rfloor \rfloor \leq \lfloor \log \lfloor \log (n-1) \rfloor \rfloor = \lfloor \log (\lceil \log n \rceil - 1) \rfloor = \lceil \log \lceil \log n \rceil \rceil - 1$ bits. The concatenation $l' = l_{-1} \cdot l_0 \cdot l_1 \cdot l_2$ now has length at most $\lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil$ bits. Finally, we prefix this with the string $l_{-2} = \mathtt{0}^j\mathtt{1}$, where $j = \lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil - |l'|$, giving the final label $l = l_{-2} \cdot l'$ of length exactly equal to $|l| = \lfloor \log n \rfloor + \lceil \log \lceil \log n \rceil \rceil + 1$, where the bits belonging to l' are those coming after the first $\mathtt{1}$. Since the length $|l|$ uniquely determines $\lfloor \log n \rfloor$, the decoder can infer $\lfloor \log n \rfloor$ from the length of the label alone, and hence it can retrieve $l_{-2} \cdot l_{-1}$ and $l_0 \cdot l_1 \cdot l_2$. From the former, it can infer l_{-1} and hence $|l_2|$, and from the latter it can infer $l_1 \cdot l_2$, and thereby also l_1 and l_2.

In the second labeling scheme for Trees, we use Lemma A.1 to encode the length $|l_2|$ as a string l_0 with $\lfloor \log |l_2| \rfloor \leq \lfloor \log \lfloor \log \delta \rfloor \rfloor$ bits. Next we encode $|l_0|$ as the string $l_1 = \mathtt{0}^{|l_0|}\mathtt{1}$ with $|l_0| + 1 \leq \lfloor \log \lfloor \log \delta \rfloor \rfloor + 1$ bits. We now create the final label $l = l_{-1} \cdot l_0 \cdot l_1 \cdot l_2$ of length at most $\lfloor \log N \rfloor + 2\lfloor \log \lfloor \log \delta \rfloor \rfloor + 1$. Since l_{-1} is

self-delimiting, the decoder can retrieve l_{-1} and $l_0 \cdot l_1 \cdot l_2$. From the former it can retrieve $|l_0|$, and from the latter it can then retrieve l_0 and $l_1 \cdot l_2$. From the former it can now retrieve $|l_2|$ and from the latter it can then retrieve l_1 and l_2. □

Chapter 5

Nearest common ancestor

The problem of finding nearest common ancestors (NCAs) is non-trivial already in the non-distributed setting. Aho, Hopcroft and Ullman [3] were among the first to consider the problem of finding NCAs, and Harel and Tarjan [25] were the first to describe an algorithm that uses only linear time and space for preprocessing and can answer NCA queries in constant time. Their algorithm is distributed for complete binary trees, but uses a non-distributed, precomputed auxiliary data structure in order to generalize the results to arbitrary trees. A more recent, much simpler, asymptotically optimal and completely distributed algorithm was presented by Alstrup et al. [6] in the form of an NCA labeling scheme. A modification to Alstrup et al.'s algorithm with better constant factors was given in [23]. The problem of finding a nearest common ancestor labeling scheme is equivalent to the *discrete range searching problem* [19]. A survey on NCAs and their applications can be found in [6].

We define the *NCA function* for any tree T and any nodes $u, v \in T$ by

$$\mathrm{nca}(u, v) = \text{the nearest common ancestor of } u \text{ and } v.$$

Note that a node is always assumed to be its own ancestor so that, in particular, $\mathrm{nca}(v, v) = v$ for all nodes v in T. We define an *NCA labeling scheme for* $\mathcal{T}$ for any subfamily $\mathcal{T} \subseteq \mathsf{Trees}$ as a labeling scheme $\sigma = (\mathrm{enc}, \mathrm{dec})$, which is the special case of the definition in Section 1.4 with $f = \mathrm{enc} \circ \mathrm{nca}$, $k = 2$ and $S = \{0, 1\}^*$, and where the encoder produces unique labels. Note that we have used the encoder as part of the definition of f, which basically means that we are defining the encoder in terms of itself. There is nothing mathematically wrong with this, however, since the requirement in (1.1) for a labeling scheme translates into the relation

$$\mathrm{enc}(\mathrm{nca}(u, v)) = \mathrm{dec}(\mathrm{enc}(u), \mathrm{enc}(v)),$$

which certainly is well-defined. So even though the encoder in an NCA labeling scheme needs to satisfy an equation where it appears on both sides of the equality sign, the definition overall does not need to pull itself up by its bootstraps. What causes the confusion is the terminology. The name "NCA labeling scheme" is ambiguous compared to the names of previous labeling schemes, because it does not uniquely describe what the function f should be; on the other hand, the name "f-labeling scheme" is not very useful since we do not want to fix a choice of f

Family	Bound on label size	Reference
$\mathsf{Trees}(N, D)$	$L \geq \lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	Theorem 5.1, [5]
$\mathsf{BinaryTrees}(N)$	$L \geq \lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	Theorem 5.1, [5]
$\mathsf{Paths}(N)$	$L \geq \lfloor \log N \rfloor$	Theorem 5.1
$\mathsf{Trees}(N)$	$L \geq \lfloor 1.008 \log N \rfloor - 317$	Theorem 5.4, [8, 7]
Trees	$\ell \leq \lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$	Theorem 5.11
BinaryTrees	$\ell \leq \lceil (1 + \log 3)\lfloor \log n \rfloor \rceil + 1$	Theorem 5.13
Caterpillars	$\ell \leq \lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil + 1$	Theorem 5.14

Table 5.1: Bounds on the optimal worst-case label size for the NCA labeling problem for different families of trees. See Section 1.7 for further detail.

Family $\mathcal{G}$	Lower bound on L for $\mathcal{G}(N)$	Upper bound on ℓ for $\mathcal{G}$
Trees	$\lfloor 1.008 \log N \rfloor - 317$	$\lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$
$\mathsf{Trees}(\Delta)$	$\lfloor 1.008 \log N \rfloor - 317$	$\lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$
$\mathsf{Trees}(D)$	$\lceil \log N \rceil + \lfloor \log \lfloor \log D \rfloor \rfloor - 1$	$\lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$
BinaryTrees	$\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$	$\lceil (1 + \log 3)\lfloor \log n \rfloor \rceil + 1$
Caterpillars	$\lfloor \log N \rfloor$	$\lfloor \log n \rfloor + \lceil \log \lfloor \log n \rfloor \rceil + 1$

Table 5.2: Some of the consequences of the results in Table 5.1.

beforehand. We shall keep the terminology "NCA labeling scheme" and leave it to the reader to remember the special definition. Table 5.1 shows some of the lower and upper bounds of the corresponding *NCA labeling problem*.

The reason why the encoder of an NCA labeling scheme is required to produce unique labels is to ensure that the output from the decoder uniquely determines a node in a given tree. If uniqueness were not required, one could give the same label to all nodes in all trees and define the decoder to return this label no matter the input, which would produce a rather pathetic NCA labeling scheme. The requirement that all nodes be their own ancestor is also quite standard when dealing with NCAs. Were this not the case, it would simply not be possible to define $\mathrm{nca}(u, v)$ in the case where either u or v is the root.

The main results of this chapter (Theorems 5.4 and 5.11 to 5.14) have been submitted [7] to the 2014 ACM-SIAM Symposium on Discrete Algorithms (SODA).

5.1 Lower bounds

The labeling schemes community has been remarkably quiet when it comes to lower bounds for the NCA labeling problem. In fact, until recently the only

known lower bound for any class of trees is the lower bound from Chapter 3 for the ancestry problem. We shall here establish a better lower bound for general trees, but we begin with some of the smaller families for which the trivial lower bound is still the best bet:

Theorem 5.1. *For any N, D with $N \geq D+1 \geq 3$, any NCA labeling scheme for* $\mathsf{Trees}(N, D)$ *has a worst-case label size of at least* $\lceil \log N \rceil + \lfloor \log(\lfloor \log D \rfloor + 1) \rfloor - 1$. *Further, for any $N \geq 2$, any NCA labeling scheme for* $\mathsf{BinaryTrees}(N)$ *has a worst-case label size of at least* $\lfloor \log N \rfloor + \lfloor \log \lfloor \log N \rfloor \rfloor - 2$. *Finally, for any N, any NCA labeling scheme for* $\mathsf{Paths}(N)$ *has a worst-case label size of at least* $\lfloor \log N \rfloor$.

Proof. The statements are immediate consequences of Theorems 3.1 to 3.3 since any NCA labeling scheme can be transformed into an ancestry labeling scheme without changing the labels. □

There is still a rather large gap from the lower bounds in the previous theorem to the upper bounds that will be established in the next section. The fact that queries in an NCA labeling scheme produce labels rather than Boolean values makes it difficult to apply traditional methods such as the boxes and groups technique from [5] (Lemma A.9). However, for trees of bounded maximum degree (and hence also for general trees), it is possible to establish a nontrivial lower bound. The remainder of this section is devoted to this task, and the lower bound we present is based on a novel technique by Alstrup and Larsen [8]. Their result is only sketched in [8], so the following is a more detailed writeup of their idea with some minor adjustments and a better constant in the final result.

5.1.1 Levenshtein distance and 3-2 sequences

We begin with some definitions. The *Levenshtein distance* [32], also known as the *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. More formally, $\mathrm{lev}(x, y)$ is defined as the number $\mathrm{lev}_{x,y}(|x|, |y|)$, where $\mathrm{lev}_{x,y}(i, j) = \max(i, j)$ whenever $\min(i, j) = 0$ and

$$\mathrm{lev}_{x,y}(i,j) = \min \begin{cases} \mathrm{lev}_{x,y}(i-1,j)+1 \\ \mathrm{lev}_{x,y}(i,j-1)+1 \\ \mathrm{lev}_{x,y}(i-1,j-1) + \begin{cases} 0, & \text{if } x_i = y_j \\ 1, & \text{otherwise} \end{cases} \end{cases}$$

whenever $\min(i, j) > 0$. Note that the first of the three choices in the large minimum corresponds to deletion of an entry in x; the second choice corresponds to deletion of an entry in y or, equivalently, insertion of an entry into x; and the third choice corresponds to substitution when the respective symbols differ.

The Levenshtein distance satisfies the properties of a metric, including symmetry and the triangle inequality. From the definition it is clear that, given two sequences x and y, there exist sequences z, w such that $\mathrm{lev}(x,y) = \mathrm{lev}(x,z) + \mathrm{lev}(z,w) + \mathrm{lev}(w,y)$ and such that z can be obtained from x by performing $\mathrm{lev}(x,z)$ deletions, w can be obtained from z by performing $\mathrm{lev}(z,w)$ substitutions, and y can be obtained from w by performing $\mathrm{lev}(w,y)$ insertions. In this sense, z and w are intermediate sequences obtained along the way while transforming x into y. Note that we will never perform more than one single-character edit on a single entry.

We define a *3-2 sequence*[1] of length $2k$ to a be an integer sequence $x = (x_1, \ldots, x_{2k})$ with exactly k 2s and k 3s.

Lemma 5.2. *For any h, k with $2 \leq h \leq k$ and k an integer with $k \geq 90$, there exists a set Σ of 3-2 sequences of length $2k$ with $|\Sigma| \geq 2^{1.95k}/(16k/h)^{3h}$ 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 $2^{1.95k}/(16k/h)^{3h} \leq 2^{1.95k}/(16k/\lfloor h \rfloor)^{3\lfloor h \rfloor}$, we can safely assume that h is an integer: for if not, replace h by $\lfloor h \rfloor$ to obtain an even stronger result.

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 [38] 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 [38], 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 $k \geq 90$. □

[1] This name has been chosen to comply with the corresponding 3-2 trees—see Section 5.1.2.

5.1.2 3-2 trees

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*[2] *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 5.3. *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]The related concept of a "2-3 tree" [4] has a slightly different definition, which is why we use a different terminology here.

2/3. It follows that the number m of leaves in T^S satisfies $m \leq 6^k \cdot (2/3)^{\mathrm{lev}(x,I(S))}$, which implies $\mathrm{lev}(x, I(S)) \leq \log_{3/2}(6^k/m)$ as desired. □

We are now ready to present the lower bound for trees with bounded maximum degree, and hence also for general trees. The result requires N to be rather large, but it is worth noting that a much smaller N would have been possible in exchange for a smaller (but still > 1) coefficient to the $\log N$ term.

Theorem 5.4. *For any $N \geq 2 \cdot 3^{240}$ and any $\Delta \geq 4$, any NCA labeling scheme for $\mathsf{Trees}(N, \Delta)$ has a worst-case label size of at least $\lfloor 1.008 \log N \rfloor - 317$.*

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 5.2, there exists a set Σ of 3-2 sequences of length $2k$ with $|\Sigma| \geq 2^{1.95k}/(16k/h)^{3h}$ and $\mathrm{lev}(x, y) > h$ for all $x, y \in \Sigma$. The set Σ defines a set of $|\Sigma|$ associated 3-2 trees with n leaves, at most $2n \leq N$ nodes and degree at most 4. In particular, all the associated trees belong to $\mathsf{Trees}(N, \Delta)$. 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 5.3, we would have

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

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

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

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

$$L \geq \lfloor 1.008 \log n \rfloor - 3 \geq \lfloor 1.008 \log(N/2 \cdot 6^{120}) \rfloor - 3 \geq \lfloor 1.008 \log N \rfloor - 317. \quad \square$$

5.2 Upper bounds

This section presents a selection of NCA labeling schemes for various families of trees. The schemes will all use the same general structure for the labels and will only differ in the way that labels are encoded. The technique uses Harel and Tarjan's *heavy-light decomposition* [25], which will be described in Section 5.2.1. Alstrup et al. [6] used this technique to create an NCA labeling scheme for general trees with labels of $O(\log n)$ bits, matching the (trivial) lower bound of $\Omega(\log n)$. A more detailed analysis in [23] revealed that the worst-case label size in their labeling scheme is $10 \log n + O(1)$ bits, and a modification in [23] allowed the label size to be decreased to $4\lfloor \log n \rfloor$ bits. We shall here presents additional modifications that decreases the worst-case label size from [23] even further to approximately $2.772 \log n$ bits for general trees and approximately $2.585 \log n$ for binary trees.

Fischer [16] ran a series of experiments to test different encodings. He measured average label lengths, worst-case label lengths, preprocessing time and query time for randomly generated families of general trees, binary trees, paths and stars. Overall, his tests revealed that non-prefix-free codes in general lead to shorter label sizes while preserving the query time. The experiments included three different encodings [26, 33, 6] that produce labels of size $O(\log n)$, and the constant factor for all three encodings was measured to reach a maximum of around 8 for binary trees. Note that these were *average* worst-case label sizes and not global worst-case label sizes. Comparing this to our results, it is clear that our new encoding strategy dramatically outperforms any of the previous.

5.2.1 Heavy-light decomposition

Heavy-light decomposition of a graph is a technique from [25] to decompose a rooted tree into paths, where each node of a path, except the top-node, has

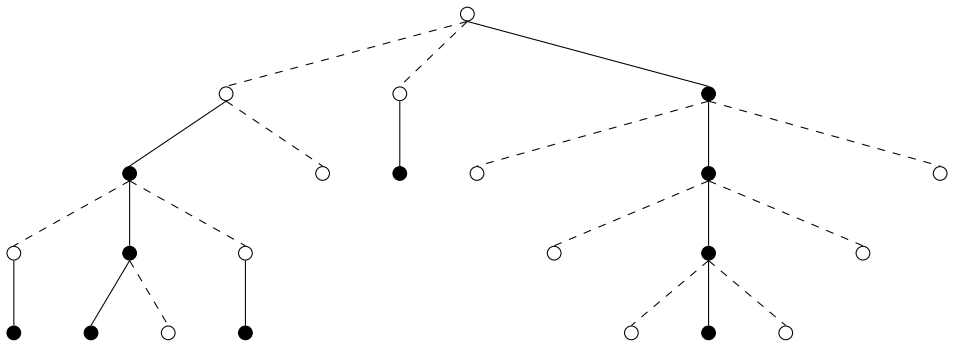


Figure 5.1: A tree in which light and heavy nodes have been marked with "○" and "●", respectively, and heavy and light edges have been drawn with solid and dashed lines, respectively.

maximum size[3] among its siblings. The decomposition allows the location of a node in the tree to be described by the sequence of paths that must be traversed in order to reach the node from the root.

More formally, let T be a tree with root r. The nodes of T are classified as either *heavy* or *light* as follows. The root is light. For each internal node $v \in T$, pick one child w whose size 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 of v by $\mathrm{lchildren}(v)$. The *light size* of a node v is the number

$$\mathrm{lsize}(v) = 1 + \sum_{w \in \mathrm{lchildren}(v)} \mathrm{size}(w).$$

Note that, if v is a leaf or has only a single child, then $\mathrm{lsize}(v) = 1$, and if v is internal, then $\mathrm{lsize}(v) = \mathrm{size}(v) - \mathrm{size}(\mathrm{hchild}(v))$. The *apex* of v, denoted $\mathrm{apex}(v)$, is the nearest light ancestor of v. An edge connecting a light node to its parent is a *light edge*, and edge connecting a heavy node to its parent is a *heavy edge*. By removing the light edges, 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 $\mathrm{apex}(v)$, which is the apex of all nodes on $\mathrm{hpath}(v)$. See Figure 5.1 for an example.

Given a node v in T, consider the list of light nodes $u_0, \ldots, u_k$ encountered on the path from the root r to v. The first of these light nodes is the root, $r = u_0$. 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. Since the size of every light node is bounded by the size of its heavy sibling, the size of u_{i+1} is at most half the size of u_i. From this it follows that $\mathrm{ldepth}(v) \leq \lfloor \log n \rfloor$ for all nodes v, where n is the number of nodes in T. See [25] for further details.

[3]See Section 0.4 in the preliminaries for a definition.

5.2.2 One NCA labeling scheme to rule them all

We now describe the labeling scheme that will be used for both Trees, BinaryTrees and Caterpillars, although with different encodings for each family. Let T be a tree with root r. We begin by assigning to each node v a *heavy label*, $\text{hlabel}(v)$, and, when v is light and not equal to the root, a *light label*, $\text{llabel}(v)$, as described in Lemmas 5.5 and 5.6 below.

Lemma 5.5. *There exist binary strings* $\text{hlabel}(v)$ *for all nodes* v *in* T *such that the following hold for all nodes* v, w *belonging to the same heavy path:*

$$\text{depth}(v) < \text{depth}(w) \implies \text{hlabel}(v) \prec \text{hlabel}(w) \tag{5.1}$$

$$|\,\text{hlabel}(v)| \leq \lfloor \log \text{size}(\text{apex}(v)) - \log \text{lsize}(v) \rfloor \tag{5.2}$$

Proof. Consider each heavy path H separately and use the sequence $(\text{lsize}(v))_{v \in H}$, ordered ascendingly by $\text{depth}(v)$, as input to Lemma A.3 from Appendix A. □

Lemma 5.6. *There exist binary strings* $\text{llabel}(v)$ *for all light nodes* $v \neq r$ *in* T *such that the following hold for all light siblings* v, w*:*

$$v \neq w \implies \text{llabel}(v) \neq \text{llabel}(w) \tag{5.3}$$

$$|\,\text{llabel}(v)| \leq \lfloor \log \text{lsize}(\text{parent}(v)) - \log \text{size}(v) \rfloor \tag{5.4}$$

Proof. Consider each set L of light siblings separately and use the sequence $(\text{size}(v))_{v \in L}$, not caring about order, as input to Lemma A.3 from Appendix A. □

In many cases we are not going to use the constructions in Lemmas 5.5 and 5.6 directly, but will instead use the modifications in Lemmas 5.7 and 5.8 below.

Lemma 5.7. *It is possible to modify the constructions in Lemmas 5.5 and 5.6 so that, for all nodes* u, v *where* v *is a light child of* u*,*

$$\text{hlabel}(u) = \varepsilon \implies \text{llabel}(v) \neq \varepsilon. \tag{5.5}$$

The modification still satisfies (5.1)*,* (5.2)*,* (5.3) *and* (5.4) *except that when* $\text{hlabel}(u)$ *is empty,* (5.4) *is replaced by*

$$|\,\text{hlabel}(u)| + |\,\text{llabel}(v)| \leq \lfloor \log \text{size}(\text{apex}(u)) - \log \text{size}(v) \rfloor \tag{5.6}$$

Proof. First observe that without modifying the construction in the two lemmas we can combine (5.2) with (5.4) to obtain (5.6). We now describe the modification: the construction works exactly as before except that in cases where $\text{hlabel}(u)$ is empty, we use Lemma A.4 in place of Lemma A.3 in the construction of light labels in Lemma 5.6. This clearly makes (5.5) true, so it remains to prove (5.6).

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 A.4. Further, $\mathrm{lsize}(u) + \mathrm{size}(\mathrm{hchild}(u)) = \mathrm{size}(u)) \leq \mathrm{size}(\mathrm{apex}(u))$. Using these two facts together, Lemma A.4 now yields

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

Since $|\,\mathrm{hlabel}(u)| = 0$, we have therefore obtained (5.6). □

Lemma 5.8. *It is possible to modify the constructions in Lemmas 5.5 and 5.6 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,*

$$\mathrm{hlabel}(u) = \varepsilon \text{ and } \mathrm{llabel}(v) = \varepsilon \implies \mathrm{hlabel}(w) \neq \varepsilon. \tag{5.7}$$

The modification still satisfies (5.1), (5.2), (5.3) *and* (5.4) *except that when* $\mathrm{hlabel}(u)$ *and* $\mathrm{llabel}(v)$ *are both empty,* (5.2) *is replaced by*

$$|\,\mathrm{hlabel}(u)| + |\,\mathrm{llabel}(v)| + |\,\mathrm{hlabel}(w)| \leq \lfloor \log \mathrm{size}(\mathrm{apex}(u)) - \log \mathrm{lsize}(w) \rfloor \tag{5.8}$$

Proof. The proof is similar to that of the previous lemma. First observe that without modifying the construction in the two lemmas we can combine (5.2), (5.4) and (5.2) again to obtain (5.8). We now describe the modification: the construction works exactly as before except that in cases where $\mathrm{hlabel}(u)$ and $\mathrm{llabel}(v)$ are both empty, we use Lemma A.4 in place of Lemma A.3 in the construction of heavy labels in Lemma 5.5. This clearly makes (5.7) true, so it remains to prove (5.8).

So let u, v and w be as above. Note that $\mathrm{size}(v)$ 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 A.4. Further, $2\,\mathrm{size}(v) \leq \mathrm{lsize}(u) + \mathrm{size}(\mathrm{hchild}(u)) = \mathrm{size}(u)) \leq \mathrm{size}(\mathrm{apex}(u))$. Using these two facts together, Lemma A.4 now yields

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

Since $|\,\mathrm{hlabel}(u)| = |\,\mathrm{llabel}(v)| = 0$, we have therefore obtained (5.8). □

It is worth noting that, if we use the modification in Lemma 5.7, then we will never be in a situation where it is possible to use the modification in Lemma 5.8.

We next assign a new set of labels for the nodes of T. Given a node v with $\mathrm{ldepth}(v) = k$, 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 $u_i = \mathrm{apex}(v_i)$ is light for $i = 0, \ldots, k$ and $v_{i-1} = \mathrm{parent}(u_i)$ for $i = 1, \ldots, k$. Let

$$l(v) = (h_0, l_1, h_1, \ldots, l_k, h_k),$$

where $l_i = \mathrm{llabel}(u_i)$ and $h_i = \mathrm{hlabel}(v_i)$.

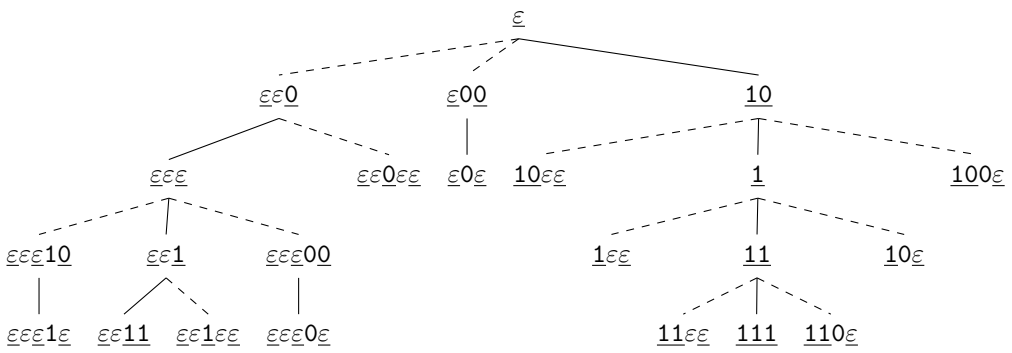


Figure 5.2: The tree from Figure 5.1 with the labels $l(v)$ with heavy sub-labels underlined.

Every node v in T now has a label $l(v)$, which is a sequence of binary strings. Figure 5.2 shows an example of a tree with these labels, where we have underlined the heavy sub-labels in order to be able to distinguish them from the light sub-labels. Note that we have used Lemmas 5.5 and 5.6 for the construction of labels in this figure and not the modifications in Lemmas 5.7 and 5.8.

To define a labeling scheme, it still remains to encode the labels $l(v)$ into a single binary string. Before we do this, however, we prove that $l(\text{nca}(v, w))$ can be computed directly from $l(v)$ and $l(w)$, which is exactly what is required by labels in an NCA labeling scheme.

Lemma 5.9. *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 $l(u) = l(v)$.

(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 (5.1) 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

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

By (5.3), $\text{llabel}(\hat{v})$ 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 (5.1), $\text{hlabel}(u) \prec \text{hlabel}(\hat{v})$, so (d) applies. The case where $\hat{w}$ is heavy is analogous. □

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, let $\tilde{l}(v)$ denote the concatenation $h_0 \cdot l_1 \cdot h_1 \cdots l_k \cdot h_k$ of the sub-labels of $l(v)$.

Lemma 5.10. *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 5.5 and 5.6 or any of the variants in Lemmas 5.7 and 5.8 for the construction of heavy and light labels.*

Proof. Let v be an arbitrary node in T and let $u_0, u_1, \dots, u_k$ denote the light nodes encountered on the path from the root $r = u_0$ to v. Set $v_k = v$ and $v_{i-1} = \text{parent}(u_i)$ for $i = 1, \dots, k$, and note that $u_i = \text{apex}(v_i)$ for all i. By construction, $l(v) = (h_0, l_1, h_1, \dots, l_k, h_k)$ where $l_i = \text{llabel}(u_i)$ and $h_i = \text{hlabel}(v_i)$.

If we use Lemmas 5.5 and 5.6 for the construction of heavy and light labels, we have by (5.2) that $|h_i| \leq \lfloor \log \text{size}(u_i) - \log \text{lsize}(v_i) \rfloor$ for all $i = 0, \dots, k$ and by (5.4) that $|l_i| \leq \lfloor \log \text{lsize}(v_{i-1}) - \log \text{size}\, u_i \rfloor$ for $i = 1, \dots, k$. Summarizing gives a telescoping sum:

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

In the cases where we have used any of the variants in Lemmas 5.7 and 5.8, we must use (5.6) or (5.8) 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. □

5.2.3 The results

From the preceding discussion we know that we can give any node v in a tree a label $l(v) = (h_0, l_1, h_1, \dots, l_k, h_k)$ consisting of $2k + 1$ sub-labels, where $k =$

$\text{ldepth}(v) \leq \lfloor \log n \rfloor$, whose concatenation $\tilde{l}(v) = h_0 \cdot l_1 \cdot h_1 \cdots l_k h_k$ has length at most $\lfloor \log n \rfloor$. If we use the modification in Lemma 5.7, then h_i and l_{i+1} cannot both be empty at the same time, and if we use the modification in Lemma 5.8 then h_i, l_{i+1} and h_{i+1} cannot all three be empty at the same time. Thanks to Lemma 5.9 the decoder in an NCA labeling scheme needs nothing more than these labels in order to be able to compute the label for the NCA of two nodes. Thus, the only thing missing is a way to encode these labels as a single binary string. Note that the decoder in an NCA labeling scheme not only has to decode such a string into the original labels but also has to re-encode the label of the NCA into such a string. Theorems 5.11 to 5.14 below use different encodings depending on the family of trees.

Theorem 5.11. *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 5.7 to ensure that every empty heavy label is followed by a nonempty light label. This means that the sequence $l(v) = (h_0, l_1, h_1, \ldots, l_k, h_k)$ can be encoded using $\lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$ bits; see Lemma A.6 in Appendix A. Given the encoded labels from two nodes, the decoder can now decode the labels as described in Lemma A.6, use Lemma 5.9 to compute the label of the NCA, and then re-encode that label using Lemma A.6 once again. □

Although time complexities are ignored throughout this thesis, it is worth noting that the above algorithm comes without any guarantees for the time complexities of encoding and decoding. Indeed, the result in Lemma A.6 is based on the *existence* of a one-to-one correspondence between the set of possible labels $l(v)$ and the set of binary strings of length at most $\lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil$ but without any concrete algorithm for how to map one set to the other and vice versa. Theorem 5.12 below is a weaker version of Theorem 5.11 but guarantees linear time encoding and constant time decoding.

Theorem 5.12. *There exists an NCA labeling scheme for* Trees *whose worst-case label size is at most* $3\lfloor \log n \rfloor$ *bits.*

Proof. The proof is identical to that of Theorem 5.11 but with Lemma A.7 in place of Lemma A.6. □

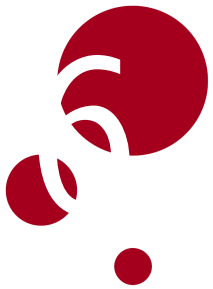
Theorem 5.13. *There exists an NCA labeling scheme for* BinaryTrees *whose worst-case label size is at most* $\lceil (1 + \log 3)\lfloor \log n \rfloor \rceil + 1 \leq 2.585 \log n + 2$.

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 modified construction in Lemma 5.8, 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 $\lceil(1+\log 3)(\lfloor\log n\rfloor-1)\rceil+3 \leq \lceil(1+\log 3)\lfloor\log n\rfloor\rceil+1$ bits; see Lemma A.8 in Appendix A. The rest of the proof follows the same argument as the proof of Theorem 5.11. □

Theorem 5.14. *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 A.5 in the appendix. 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 by the first bit. The rest of the proof follows the same argument as the proof of Theorem 5.11. □



Chapter 6

Distance

Graham and Pollak [22] proposed to embed graphs in so-called *squashed cubes*, thereby allowing the distance between two nodes to be computed as the Hamming distance [24] between some naturally defined labels for the faces of the squashed cube. They conjectured the smallest dimension of such a squashed cube, and their conjecture was subsequently proven by Winkler [43]. This was the first example of a distance labeling scheme for graphs, although the general concept of labeling schemes had not yet been defined. More recently, Peleg [37] noticed that the labeling scheme concept introduced for adjacency in [27] could also be used for distances in trees. This and other similar observations led to the birth of the modern definition of general labeling schemes.

We define the *distance function* for any graph G and any nodes $u, v \in G$ by

$$\text{dist}(u, v) = \text{the distance between } u \text{ and } v.$$

Note that $\text{dist}(u, v) = 0$ if and only if $u = v$; $\text{dist}(u, v) = 1$ if and only if u and v are adjacent; and $\text{dist}(u, v) = \infty$ if and only if u and v are not connected. A *distance labeling scheme for* $\mathcal{G}$ is defined for any subfamily $\mathcal{G} \subseteq \mathsf{Graphs}$ as the special case of the definition in Section 1.4 with $f = \text{adj}$, $k = 2$ and $S = \mathbb{N}_0 \cup \{\infty\}$. Table 6.1 shows some of the lower and upper bounds of the corresponding *distance labeling problem*. Since a node has distance 0 only to itself, a distance labeling scheme must produce unique labels.

The primary focus of this thesis is on trees, but distances can be defined for general graphs, too, and we therefore extend the discussion in this chapter to also cover graphs, although without going into details.

6.1 Lower bounds

The best known lower bound for distance labeling schemes for general graphs is somewhat disappointing: it is just a copy of the lower bound for adjacency for general graphs. Whether this is because an optimal distance labeling scheme truly has labels of the same size as an optimal adjacency labeling scheme, or if it is just because a better bound has not yet been established is hard to say. Gavoille et al. [21] establish lower bounds for distance labeling schemes for a whole collection of graph families in both the exact and approximate cases, but

Family	Bound on label size	Reference
$\mathsf{Graphs}(N)$	$L \geq \lfloor \frac{1}{2}N \rfloor - 1$	Theorem 6.1, [34]
$\mathsf{Trees}(N, D)$	$L \geq \lfloor \log N \rfloor$	Theorem 6.2
$\mathsf{BinaryTrees}(N)$	$L \geq \lfloor \frac{1}{2} \lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	Theorem 6.5, [21]
$\mathsf{Caterpillars}(N)$	$L \geq 2\lfloor \log N \rfloor - \lfloor \log \lfloor \log N \rfloor \rfloor - 4$	Theorem 6.6
Graphs	$\ell \leq \lfloor (\log 3)n \rfloor + \lceil \log n \rceil$	Theorem 6.7, [43]
Trees	$\ell \leq \lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$	Theorem 6.8
$\mathsf{BinaryTrees}$	$\ell \leq \lceil \frac{1}{2}\lceil \log n \rceil^2 + (\frac{3}{2} + \log 3)\lceil \log n \rceil \rceil + 1$	Theorem 6.9
$\mathsf{Caterpillars}$	$\ell \leq 2\lfloor \log n \rfloor$	Theorem 6.10

Table 6.1: Bounds on the optimal worst-case label size for the distance labeling problem for different families of graphs. See Section 1.7 for further detail.

Family $\mathcal{G}$	Lower bound on L for $\mathcal{G}(N)$	Upper bound on ℓ for $\mathcal{G}$
Graphs	$\lfloor \frac{1}{2}N \rfloor - 1$	$\lfloor (\log 3)n \rfloor + \lceil \log n \rceil$
Trees	$\lfloor \frac{1}{2} \lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$
$\mathsf{Trees}(\Delta)$	$\lfloor \frac{1}{2} \lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$
$\mathsf{Trees}(D)$	$\lfloor \log N \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil$
$\mathsf{BinaryTrees}$	$\lfloor \frac{1}{2} \lfloor \frac{1}{2}(\log N - 1) \rfloor^2 \rfloor$	$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (\frac{3}{2} + \log 3)\lceil \log n \rceil \rceil + 1$
$\mathsf{Caterpillars}$	$2\lfloor \log N \rfloor - \lfloor \log \lfloor \log N \rfloor \rfloor - 4$	$2\lfloor \log n \rfloor$

Table 6.2: Some of the consequences of the results in Table 6.1.

when applied to Graphs in the exact case, their lower bound is identical to the one we have already established for adjacency.

Theorem 6.1. *For any N, any distance labeling scheme for $\mathsf{Graphs}(N)$ has a worst-case label size of at least $\lfloor \frac{1}{2}N \rfloor - 1$.*

Proof. Follows from Theorem 2.1, since any distance labeling scheme can be transformed into an adjacency labeling scheme without changing the labels. □

Moving on to trees, we begin with a trivial result for trees of bounded depth, where we simply exploit that labels in a distance labeling scheme must be unique.

Theorem 6.2. *For any N, D, any distance labeling scheme for $\mathsf{Trees}(N, D)$ has a worst-case label size of at least $\lfloor \log N \rfloor$.*

In the case of general trees, Gavoille et al [21] establish a lower bound using an ingenious technique where they apply a distance labeling scheme to a special class of trees called (h, M)-trees. Unfortunately, their proof has some errors and unclarities, which we shall try to fix in the following exposition. We shall later describe in more detail the problems with their proof.

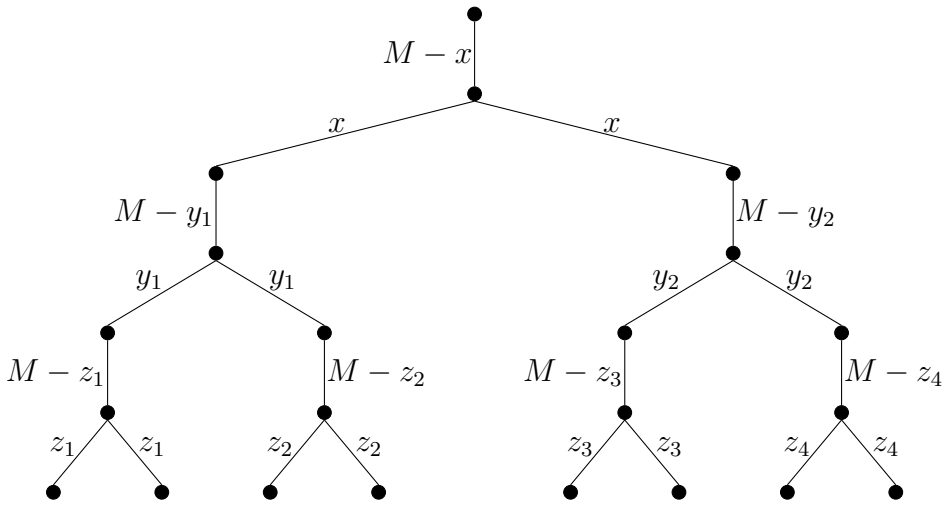


Figure 6.1: An (h, M)-tree, where $h = 3$. Each edge should be replaced by a path that is as long as the edge label indicates. If an edge label is 0, then the two end nodes collapse into one single node.

6.1.1 (h, M)-trees

We begin with some definitions. For $h \geq 0$ and $M \geq 2$, an (h, M)-*tree* is a binary tree that is constructed recursively as follows. For $h = 0$, the tree is just a single node. For $h > 0$, the tree consists of a path of length $M - x$ for some $0 \leq x < M$ whose one end point is the root of the tree and whose other end point is the start node of two paths of length x whose end points are the roots of two $(h - 1, M)$-trees. An example for $h = 3$ can be seen in Figure 6.1. We shall denote an (h, M)-tree constructed in this way by $T = \langle T_0, T_1, x \rangle$, where T_0 and T_1 are the two $(h - 1, M)$-trees attached to the ends of the two paths of length x. Note that, if $x = 0$, then the two paths of length 0 collapse into a single node, and the two subtrees T_0 and T_1 are then both attached to this node.

It is easy to see that an (h, M)-tree has at most 2^h leaves, at most $2^{h+1}M$ nodes in total and depth exactly equal to hM. Further, it is straightforward to see that, if u, v are leaves in an (h, M)-tree $T = \langle T_0, T_1, x \rangle$, then

$$\mathrm{dist}_T(u, v) = \begin{cases} 2(h-1)M + 2x, & \text{if } u \in T_0 \text{ and } v \in T_1, \text{ or vice versa,} \\ \mathrm{dist}_{T_i}(u, v), & \text{if } u, v \in T_i \text{ for some } i = 0, 1. \end{cases} \tag{6.1}$$

6.1.2 Leaf distance labeling schemes

In the following we shall consider *leaf distance labeling schemes* for the family of (h, M)-trees: that is, distance labeling schemes where only the leaves in a tree need to be labeled, and where only leaf labels can be given as input to the decoder. Since an ordinary distance labeling scheme obviously can be used only for leaves,

any lower bound on worst-case label sizes for a leaf distance labeling scheme is also a lower bound for an ordinary distance labeling scheme. We denote by $g(h, M)$ the smallest number of labels needed by an optimal leaf distance labeling scheme to label all (h, M)-trees.

Lemma 6.3. *For all $h \geq 1$ and $M \geq 2$, $g(h, M)^2 \geq Mg(h-1, M^2)$.*

Proof. Let h and M be given, and fix an optimal leaf distance labeling scheme σ which produces exactly $g(h, M)$ distinct labels for the family of (h, M)-trees. For leaves u and v in an (h, M)-tree, denote by $l(u)$ and $l(v)$, respectively, the labels assigned by the scheme σ. For $x = 0, \ldots, M-1$, let $W(x)$ be the set consisting of pairs of labels $(l(u), l(v))$ for all leaves $u \in T_0$ and $v \in T_1$ in all (h, M)-trees $T = \langle T_0, T_1, x\rangle$.

The sets $W(x)$ and $W(x')$ are disjoint for $x \neq x'$, since every pair of labels in $W(x)$ uniquely determines x because of the formula in (6.1). Letting $W = \bigcup_{x=0}^{M-1} W(x)$, we therefore have $|W| = \sum_{x=0}^{M-1} |W(x)|$. Since W contains pairs of labels produced by σ from leaves in (h, M)-trees , we clearly also have $|W| \leq g(h, M)^2$, and hence it only remains to prove that $|W| \geq Mg(h-1, M^2)$, which we shall do by showing that $|W(x)| \geq g(h-1, M^2)$.

The goal for the rest of the proof is therefore to create a leaf distance labeling scheme for $(h-1, M^2)$-trees using only labels from the set $W(x)$ for some fixed x. So let x be given and consider an $(h-1, M^2)$-tree T'. From T' we shall construct an $(h-1, M)$-tree $\varphi_i(T')$ for $i = 0, 1$ such that every leaf node v in T' corresponds to a node $\varphi_i(v)$ in $\varphi_i(T')$ for $i = 0, 1$. The trees $\varphi_i(T')$ are defined as follows. If $h = 1$, so that T' consists of a single node, then $\varphi_i(T') = T'$ for $i = 0, 1$. If $h > 1$, then T' is in the form $T' = \langle T'_0, T'_1, w\rangle$ for some $0 \leq w < M^2$. We can write w in the form $w = w_0 + w_1 M$ for uniquely determined w_0, w_1 with $0 \leq w_0, w_1 < M$. For $i = 0, 1$, we recursively define $\varphi_i(T') = \langle \varphi_i(T'_0), \varphi_i(T'_1), w_i\rangle$. Thus, $\varphi_i(T')$ is an $(h-1, M)$-tree that is similar to T' but where we replace the two top paths of length w by two paths of length w_i and, recursively, do the same for all $(h-2, M^2)$-subtrees. Note also that the corresponding path of length $M^2 - w$ in T' automatically must be replaced by a path of length $M - w_i$ in $\varphi_i(T')$ in order for $\varphi_i(T')$ to be an $(h-1, M)$-tree. Note that the number of leaves in $\varphi_i(T')$ is less than or equal to the number of leaves in T' (two leaves may collapse to one if $w > 0$ and $w_i = 0$), but no matter what, any leaf in T' corresponds to a leaf in $\varphi_i(T')$, namely the leaf at the end of the corresponding path. We denote by $\varphi_i(v)$ the leaf in $\varphi_i(T')$ corresponding to the leaf v in T'.

Consider now the (h, M)-tree $T = \langle \varphi_0(T'), \varphi_1(T'), x\rangle$. Every leaf v in T' corresponds to the leaves $\varphi_0(v), \varphi_1(v)$ in T where $\varphi_i(v) \in \varphi_i(T')$ for $i = 0, 1$. Using formula (6.1) for the distances in T', it is straightforward to see that

$$\begin{aligned}\operatorname{dist}_{T'}(u, v) = \big(\operatorname{dist}_{\varphi_0(T')}(\varphi_0(u), \varphi_0(v)) \bmod (2M)\big) \\ + M \operatorname{dist}_{\varphi_1(T')}(\varphi_1(u), \varphi_1(v)). \quad (6.2)\end{aligned}$$

We can now apply the leaf distance labeling scheme σ to T and obtain a label for each leaf node in T. In particular, the pair of leaves $(\varphi_0(v), \varphi_1(v))$ corresponding to a node v in T' will receive a pair of labels. We use this pair to label v in T', whereby we have obtained a labeling of the leaves in T' with labels from $W(x)$. Using the formula in (6.2) we can construct a decoder that can compute the distance between two nodes in T' using these labels alone, and hence we have obtained a leaf distance labeling scheme for $(h-1, M^2)$-trees using only labels from $W(x)$ as desired. □

Lemma 6.4. *For all $h \geq 1$ and $M \geq 2$, $g(h, M) \geq M^{h/2}$.*

Proof. The proof is by induction on h. For $h = 1$ we note that an $(0, M)$-tree has only one node, so that $g(0, M^2) = 1$. Lemma 6.3 therefore yields $g(1, M)^2 \geq M$ from which it follows that $g(1, M) \geq \sqrt{M}$. The claim therefore holds for $h = 1$. Now let $h > 1$ and assume that the claim holds for $h - 1$. Lemma 6.3 and the induction hypothesis now yields $g(h, M)^2 \geq Mg(h-1, M^2) \geq M \cdot (M^2)^{(h-1)/2} = M^h$ from which it follows that $g(h, M) \geq M^{h/2}$. □

Theorem 6.5. *For any $N \geq 8$, any distance labeling scheme for* $\mathsf{BinaryTrees}(N)$ *has a worst-case label size of at least $\lfloor \frac{1}{2} \lfloor \frac{1}{2} (\log N - 1) \rfloor^2 \rfloor \geq \frac{1}{8} \log^2 N - \frac{3}{2} \log N$.*

Proof. Set $n = 2^{2\lfloor \frac{1}{2}(\log N - 1) \rfloor}$ be $N/2$ rounded down to the nearest even power of 2, and set $h = \log \sqrt{n}$ and $M = \sqrt{n}$. Observe that h and M are both positive integers. Any (h, M)-tree has at most $2^{h+1}M = 2\sqrt{n}\sqrt{n} \leq N$ nodes and hence belongs to $\mathsf{BinaryTrees}(N)$. A distance labeling scheme for $\mathsf{BinaryTrees}(N)$ therefore induces a leaf distance labeling scheme for (h, M)-trees, and it follows from Lemma 6.4 that such a scheme must use at least $g(h, M) \geq M^{h/2} = n^{\frac{1}{8} \log n}$ labels. If the worst-case label size is L, we can create $2^{L+1} - 1$ distinct labels, and we must therefore have $n^{\frac{1}{8} \log n} \leq 2^{L+1} - 1$ from which it follows that $L \geq \lfloor \frac{1}{8} \log^2 n \rfloor = \lfloor \frac{1}{2} \lfloor \frac{1}{2} (\log N - 1) \rfloor^2 \rfloor$. □

Note that, since the lower bound holds for binary trees, it holds, in particular, for $\mathsf{Trees}(\Delta)$ for all $\Delta \geq 3$.

6.1.3 Problems with the original proof

As mentioned previously, the original proof of Theorem 6.5 by Gavoille et al. [21] has a few problems. These include

- their construction of an $(h-1, M)$-tree from an (h, M^2)-tree (equivalent to Lemma 6.3) is not, in fact, an (h, M)-tree since the corresponding weights (number of edges) do not add up to M;
- if changing their construction to make the weights (number of edges) add up to M, their distance formula (equivalent to (6.2)) no longer holds; and

- they consider general labeling schemes and not just *leaf* labeling schemes but, nonetheless, only describe labels for the leaves without explaining how a leaf labeling scheme can be extended to a labeling scheme for the entire tree.

I contacted the corresponding author to ask him about the problems, and he replied that he and his co-authors indeed had discovered some issues with their proof after its publication, but that the issues were all fixable to the best of their knowledge. He said he would get back to me with further detail on the matter, but at the time of writing, I have still not heard anything. Thus, all improvements of their original proofs have been made by me.

6.1.4 A new technique for distances in caterpillars

The last graph family we consider is $\mathsf{Caterpillars}(N)$. We present a novel technique, a contribution of this thesis, that counts tuples of labels that are known to be distinct and compares the result to the number of tuples one can obtain with labels of size L. The technique may have applications to distance labeling for other families, and perhaps, after some minor adjustments, also to NCA or other labeling problems as well.

Theorem 6.6. *For any $N \geq 4$, any distance labeling scheme for $\mathsf{Caterpillars}(N)$ has a worst-case label size of at least $2\lfloor \log N \rfloor - \lfloor \log \lfloor \log N \rfloor \rfloor - 4$.*

Proof. Let $n = 2^{\lfloor \log N \rfloor}$ be N rounded down to the nearest power of 2, and set $k = \log n$. Let $(i_1, \ldots, i_k)$ be a sequence of k numbers from the set $\{1, \ldots, n/2\}$ with the only requirement being that $i_1 = 1$. Now consider, for each such sequence, the caterpillar whose main path has length $n/2$ and where, for $t = 1, \ldots, k$, the node in position i_t has $\lfloor n/2k \rfloor$ leaf children (not on the main path). We shall refer to these children as the *t'th group*. Note that two disjoint groups of children may be children of the same node if $i_t = i_s$ for some s, t. Each caterpillar has $n/2 + k\lfloor n/2k \rfloor \leq n \leq N$ nodes and hence belongs to $\mathsf{Caterpillars}(N)$.

Suppose that σ is a distance labeling scheme for $\mathsf{Caterpillars}(N)$, and consider one of the caterpillars defined above. Given distinct nodes u, v not on the main path, their distance will be $\mathrm{dist}(u, v) = |i_s - i_t| + 2$, where i_s and i_t are the positions on the main path of the parents of u and v, respectively. In particular, if $s = 1$, so that $i_s = 1$, then $\mathrm{dist}(u, v) = i_t + 1$. Thus, if σ has been used to label the nodes of the caterpillar, the number i_t for a child in the t'th group can be uniquely determined from its label together with the label of any of the children from the first group. It follows that any k-tuple of labels $(l_1, \ldots, l_k)$ where l_t is a label of a child in the t'th group uniquely determines the sequence $(i_1, \ldots, i_k)$. In particular, k-tuples of labels from distinct caterpillars must be distinct. Of course, k-tuples of labels from the same caterpillar must also be distinct, since labels are unique in a distance labeling scheme.

Now, there are $(n/2)^{k-1}$ choices for the sequence $(i_1, \ldots, i_k)$, and hence there are $(n/2)^{k-1}$ different caterpillars in this form. For each of these, there are $\lfloor n/2k \rfloor^k$ different choices of k-tuples of labels. Altogether, we therefore have $(n/2)^{k-1}\lfloor n/2k \rfloor^k$ distinct k-tuples of labels. If the worst-case label size of σ is L, then we can create $(2^{L+1}-1)^k$ distinct k-tuples of labels, so we must have $(n/2)^{k-1}\lfloor n/2k \rfloor^k \leq (2^L - 1)^k$. From this it follow that

$$\begin{aligned} L &\geq \lfloor \frac{k-1}{k}(\log n - 1) + \log\lfloor n/2k \rfloor \rfloor \\ &\geq \lfloor \frac{(k-1)^2}{k} + k - \log k \rfloor - 2 \\ &\geq 2k - \lfloor \log k \rfloor - 4 \\ &= 2\lfloor \log N \rfloor - \lfloor \log \lfloor \log N \rfloor \rfloor - 4. \end{aligned}$$

□

6.2 Upper bounds

The *squashed cube conjecture*, proposed by Graham and Pollak [22] and solved by Winkler [43], states that any connected graph with n nodes can be labeled using labels of length $n-1$ from the alphabet $\{0, 1, *\}$ in such a way that the Hamming distance [24] between to labels (where the distance from $*$ to both 0 and 1 is set to 0) is the same as the distance between the corresponding nodes in the graph. In particular, the conjecture states that there is a distance labeling scheme for connected graphs with labels of size $n-1$ from a 3-letter alphabet. We can encode strings from a 3-letter alphabet into binary strings by considering each string a number in base 3 and then changing base to binary, giving a binary string of length at most $\lfloor (\log 3)(n-1) \rfloor + 1 \leq \lfloor (\log 3)n \rfloor$. By padding with 0s, we can ensure that the resulting binary string has length exactly $\lfloor (\log 3)n \rfloor$. To also allow unconnected graphs, we can number the components and add as a prefix to each label the number of the corresponding component. Padding these numbers with 0s, we can do this with exactly $\lceil \log n \rceil$ bits, giving a total label size of $\lfloor (\log 3)n \rfloor + \lceil \log n \rceil$. Since $\lceil \log n \rceil$ can be uniquely determined from this number, the decoder will know which part of the label is the component number and which is the label. Thus, given two labels, it can detect if the nodes are in the same component and return ∞ if they are not and their distance if they are. This establishes the following theorem.

Theorem 6.7. *There exists a distance labeling scheme for* Graphs *whose worst-case label size is at most* $\lfloor (\log 3)n \rfloor + \lceil \log n \rceil$.

It is worth noting that Winkler's [43] solution has a quite prohibitive $\Theta(n)$ decoding time. Gavoille et al. [21] have proposed a different distance labeling scheme for graphs with a worst-case label size of $11n + O(\log n \log\log n)$ and a $O(\log\log n)$ decoding time. Weinmann and Peleg [41] later modified this algorithm so that the decoding time was further reduced to $O(\log^* n)$ with slightly

larger labels, although still of size $O(n)$. Since this thesis is only concerned with label sizes and not decoding time, we keep the result in Theorem 6.7 as the state-of-the-art distance labeling scheme for Graphs.

We now proceed to general trees. Gavoille et al. [21] prove, in a more general setting, an upper bound of $O(\log^2 n)$ for this family, which matches the $O(\log^2 n)$ lower bound. The constant factor for the lower bound (which was also established in [21]) is $1/8$, but the constant factor for the upper bound in [21] turns out to be $1/(\log 3 - 1) \approx 1.710$. We here present a distance labeling scheme for trees, a contribution of this thesis, where the constant factor is $1/2$. The scheme is an extension of the NCA labeling scheme from Theorem 5.11.

Theorem 6.8. *There exists a distance labeling scheme for* Trees *whose worst-case label size is at most* $\lceil \frac{1}{2}\lceil \log n \rceil \rceil^2 + (2 + \log(\sqrt{2}+1))\lceil \log n \rceil \rceil \leq \frac{1}{2}\log^2 n + 4.272 \log n + 5$.

Proof. Recall from Theorem 5.11 the NCA labeling scheme for Trees with a worst-case label size of exactly $\lceil (1 + \log(2+\sqrt{2}))\lfloor \log n \rfloor \rceil$. The scheme uses the heavy-light decomposition for trees and labels each node v in a tree with a sequence $(h_0, l_1, h_1, \ldots, l_k, h_k)$ of heavy and light labels, where the light labels l_i correspond to the light nodes encountered on the path from (but not including) the root to v, and each heavy label h_i labels the parent of the light node with light label l_{i+1}. We refer to Section 5.2 for definitions and notation.

Observe that the path between two nodes v and w in a tree must go through $u = \mathrm{nca}(v, w)$, and that the distance between v and w therefore can be computed as $\mathrm{dist}(v, w) = \mathrm{dist}(v, u) + \mathrm{dist}(u, w)$. We shall prove below that if each node stores its NCA label together with the distances to all its light ancestors, then it is possible for the decoder to compute these distances.

By construction, $z = \mathrm{apex}(u)$ is a light ancestor of both v and w. Note that we have

$$\mathrm{dist}(v, w) = \mathrm{dist}(v, z) + \mathrm{dist}(w, z) - 2\,\mathrm{dist}(u, z). \tag{6.3}$$

If v is an ancestor of w, then $u = v$, and (6.3) therefore yields

$$\mathrm{dist}(v, w) = \mathrm{dist}(w, z) - \mathrm{dist}(v, z),$$

which is a formula for $\mathrm{dist}(v, w)$ written in terms of distances from v, w to a common light ancestor. An analogous formula can be obtained when w is an ancestor of v. If v and w are not ancestors of each other, then u must have two distinct children $\hat{v}$ and $\hat{w}$ such that $\hat{v}$ is an ancestor of v and $\hat{w}$ is an ancestor of w. Note that $\mathrm{dist}(\hat{v}, z) = \mathrm{dist}(\hat{w}, z) = \mathrm{dist}(u, z) + 1$. At least one of $\hat{v}$ and $\hat{w}$ must be light. Supposing that it is $\hat{v}$, we then obtain from (6.3) that

$$\begin{aligned} \mathrm{dist}(v, w) &= \mathrm{dist}(v, z) + \mathrm{dist}(w, z) - 2(\mathrm{dist}(\hat{v}, z) - 1) \\ &= 2\,\mathrm{dist}(v, \hat{v}) - \mathrm{dist}(v, z) + \mathrm{dist}(w, z) + 2, \end{aligned}$$

which once again is a formula for $\text{dist}(v, w)$ written in terms of distances from v, w to light ancestors. An analogous formula can be obtained if $\hat{w}$ is light. This proves that all distances can be written in terms of distances to light ancestors.

We will now construct a distance labeling scheme for Trees. The idea is to let each node be given a label that contains its NCA label together with a list of distances to its light ancestors. Given two labels, the decoder first uses the NCA label to determine the NCA of the two nodes. By looking up entries in the lists of distances to light ancestors for the two nodes, it can then use the above formulas to compute the distance between the nodes. It only remains to find an efficient way to encode the NCA label and the list of distances into a single label.

As seen in Theorem 5.11 the NCA label uses exactly $\lceil (1+\log(2+\sqrt{2}))\lfloor \log n \rfloor \rceil$ bits. The distance from a node v to the root (which is the first light ancestor in the list) is at most $n-1$ and can therefore be encoded with *exactly* $\lceil \log n \rceil$ bits (see Lemma A.2 from Appendix A). By construction of the heavy-light decomposition, the next light node on the path to v will be the root of a subtree of size at most $n/2$, meaning that the distance from v to that ancestor is at most $n/2 - 1$ and can be encoded with *exactly* $\lceil \log n \rceil - 1$ bits. Continuing this way, we encode the i'th light ancestor with exactly $\lceil \log n \rceil - i$ bits. When we run out of light ancestors, we concatenate all the all the encoded distances, resulting in a string of length at most

$$\lceil \log n \rceil + (\lceil \log n \rceil - 1) + \cdots + 2 + 1 = \frac{1}{2}\lceil \log n \rceil^2 + \frac{1}{2}\lceil \log n \rceil.$$

By appending `0`s at the end, we can ensure that the string has *exactly* this length. Prepending the concatenated distances with the NCA label now gives a label of length exactly

$$\frac{1}{2}\lceil \log n \rceil^2 + \frac{1}{2}\lceil \log n \rceil + \lceil (1 + \log(2 + \sqrt{2}))\lfloor \log n \rfloor \rceil \tag{6.4}$$

This length is bounded by

$$\lceil \frac{1}{2}\lceil \log n \rceil^2 + (\frac{3}{2} + \log(2 + \sqrt{2}))\lceil \log n \rceil \rceil$$
$$= \lceil \frac{1}{2}\lceil \log n \rceil^2 + (2 + \log(\sqrt{2} + 1))\lceil \log n \rceil \rceil$$

Since the label length in (6.4) uniquely determines $\lfloor \log n \rfloor$, the decoder is able to split up the label into its NCA label and the entries in the list. From the NCA label it will know which entries in the list are just added `0`s that can be discarded. □

It is interesting to note that, since the above labeling scheme is able to handle both distance queries and NCA queries, it is actually also able to handle adjacency, ancestry and sibling queries as well (see the discussion at the end of

Section 1.5). Thus, this is an all-in-one labeling scheme. Although this may seem like a good thing, it is probably also an indication that this labeling scheme is suboptimal: it is stronger than need be.

If we had used Theorem 5.12 instead of Theorem 5.11 for the construction in Theorem 6.8, we would have obtained a more time efficient distance labeling scheme with only slightly larger labels, which would probably be a lot more applicable in real life. Using instead Theorem 5.13, which is specialized to binary trees, also only produce slightly smaller labels:

Theorem 6.9. *There exists a distance labeling scheme for* BinaryTrees *whose worst-case label size is at most* $\lceil \frac{1}{2}\lceil \log n \rceil^2 + (\frac{3}{2} + \log 3)\lceil \log n \rceil \rceil + 1 \leq \frac{1}{2}\log^2 n + 4.085 \log n + 6$.

Proof. The construction is similar to the one in the previous theorem except that we can use the NCA labeling scheme from Theorem 5.13 of size $\lceil (1 + \log 3)\lfloor \log n \rfloor \rceil + 1$ instead of the one from Theorem 5.11. □

Because of their simple structure, caterpillars allow for a very simple distance labeling scheme whose label size matches the lower bound from Theorem 6.6 in the dominating term:

Theorem 6.10. *There exists a distance labeling scheme for* Caterpillars *whose worst-case label size is at most* $2\lfloor \log n \rfloor$.

Proof. Given a caterpillar with n nodes, begin by enumerating all n nodes with the numbers $0, \ldots, n-1$, starting with the root node and going down the main path first before proceeding to the remaining nodes. Thus, the nodes on the main path will have numbers $0, \ldots, m-1$ and the remaining nodes will have numbers $m, \ldots, n-1$, where m is the number of nodes on the main path. We now label each node with a pair of numbers, the first being its number from the enumeration process and the second being the number of its nearest ancestor on the main path. Thus, a node on the main path will receive a label in the form (x, x) where x is its number, and a node not on the main path will receive a label in the form (x, y) where x is its number, y is the number of its parent and $x > y$. We can encode each number with $\lfloor \log n \rfloor$ bits (see Lemma A.1 in Appendix A) and each label can therefore be encoded with exactly $2\lfloor \log n \rfloor$ from which the two numbers can be uniquely determined.

Now, given the labels of two nodes u, v, the decoder can retrieve their corresponding pairs of numbers (x_u, y_u) and (x_v, y_v), respectively, and their distance

can then be computed as

$$\operatorname{dist}(u,v) = \begin{cases} 0, & \text{if } (x_u, x_v) = (y_u, y_v), \\ |x_u - x_v|, & \text{if } x_u = y_u \text{ and } x_v = y_v, \\ |x_u - y_v| + 1, & \text{if } x_u = y_u \text{ and } x_v \neq y_v, \\ |y_u - x_v| + 1, & \text{if } x_u \neq y_u \text{ and } x_v = y_v, \\ |y_u - y_v| + 2, & \text{if } x_u \neq y_u,\ x_v \neq y_v \text{ and } (x_u, x_v) \neq (y_u, y_v). \end{cases} \quad \square$$

Chapter 7

Future research

Table 1.2 on page 15 shows that there are still plenty of open problems in the world of labeling schemes. Out of the many combinations of labeling problems and graph families encountered in this thesis, only a few can rightfully claim to have been closed. Some problems are "closed" in the sense that the lower and upper bounds match in the dominating term, but before we can talk about real closure, the lower and upper bounds must meet, at least up to a small additive constant. Overall, the field of labeling schemes is still wide open.

This thesis contains a broad selection of results. Some of these establish bounds that are tighter than those previously known, while others establish bounds for families of trees that have simply not been considered before. Future research is likely to continue along the same lines, establishing even tighter bounds for even more families of trees and graphs, and of course not only for the five labeling problems considered in this thesis but for *all* labeling problems.

Labeling schemes are not just of academic interest but have many real-life applications. However, the definition used in this thesis may be too restricted and inflexible for some practical uses. Some of the variations from Section 1.6 may be closer to what is actually needed in practice, for example

- *dynamic*, since most structures change over time;
- *approximative*, since approximate results are sufficient in many situations;
- *probabilistic*, since a high probability of a correct result is sufficient in many situations; and
- *multiple query types*, since there can be more than one type of information that is of interest.

Nevertheless, these, as well as all the other variations presented in Section 1.6, seem heavily underdeveloped in the literature and are just waiting to be further explored. The number of possible variations is huge, so it could be worthwhile for future researchers to investigate how labeling schemes are used in real life before choosing which variations to consider. Inspiration from the real world may also lead to entirely new labeling problems.

The discussion in Appendix B clears up some of the confusion regarding what the decoder of a labeling scheme knows or does not know and how to best present

label sizes. Notation and terminology of this thesis is certainly more precise than a substantial part of the literature, but it is still not ideal, nor is it necessarily in line with what is desired by actual users of labeling schemes. The discussion goes on—perhaps forever.

One of the main objectives for future research should be to shed light over the fundamental labeling scheme problems, most of which, as we have seen, are still open. These problems are difficult, but future researchers can perhaps find inspiration in some of the simpler problems first. This thesis leaves behind some "low-hanging fruits" that hopefully may contribute to future progress:

- *The NCA labeling problem for trees.* During the work on this thesis, the upper and lower bounds have moved closer and closer together every few months or so. Things are certainly already happening in this area!

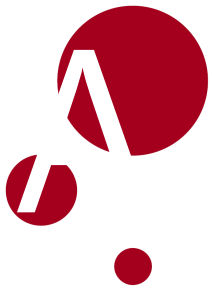
- *The NCA labeling problem for trees of bounded depth and bounded degree.* For binary trees and caterpillars, we found even smaller upper bounds than the one for general trees by using a more efficient encoding. This thesis did not investigate NCA for bounded depth and bounded degree trees, but it seems likely that something similar should be possible for these important families of trees.

- *The distance labeling problem for caterpillars.* This thesis introduced a new lower bound technique that established that distance labeling schemes for caterpillars have labels of size $2 \log n - \epsilon \log \log n$ for some $0 \leq \epsilon \leq 1$. Now it just remains to find ϵ. The new technique for caterpillars may have other applications, too.

- *The distance labeling problem for trees of bounded depth.* If the depth of a tree is bounded, so are all distances in the tree. This thesis did not investigate distance for bounded depth trees, but there are probably many straightforward results to obtain.

- *The framework.* As mentioned above, results have here been presented with larger precision and less ambiguity than is normally found in the literature, but there is still a long way to go before the ideal, if it even exists, has been found. This should certainly be investigated further, and investigations should consider how labeling schemes are used in practice.

The main objective of this thesis was to create the comparative overview in Table 1.2. The table provides many insights into the nature of labeling schemes and is a good starting point for someone unfamiliar with the subject. Future research will allow the table to be expanded to include more labeling problems, more graph families and a lot more variations, and to present increasingly tighter bounds on label sizes. The table—and this thesis—are just the beginning.



Appendix A

Efficient encodings and a lower bound technique

This chapter presents some general techniques that are not specialized to labeling schemes but can be applied in many different contexts. The first section is concerned with how to efficiently encode information using as little space as possible. The second section presents a counting technique that is useful for establishing lower bounds on label sizes. To facilitate future references, we present even fairly trivial results as explicitly stated lemmas.

A.1 Encodings

Throughout this section, a *label* is just a string assigned to an object, for example a node in a graph. For simplicity we shall only consider binary strings. A labeling of a set X of objects corresponds to map $X \to \{0,1\}^*$. If this map is injective, the labels are *unique*.

A.1.1 Labels

Lemma A.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 A.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 A.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 $L \geq \lceil \log n \rceil$. □

The labeling in Lemma A.1 corresponds to an enumeration of the n objects with numbers $1, \ldots, n$ where each number is written in base 2 and the leading 1

is removed. The labeling in Lemma A.2 corresponds to an enumeration of the n objects with numbers $0, \ldots, n-1$ where each number is written in base 2 and padded with leading 0s to obtain numbers of identical lengths. In both cases, a time efficient implementation depends entirely on whether there exists a time efficient implementation of the 1-1 correspondence between numbers and objects.

A.1.2 Ordered labels

If the objects being labeled are ordered, we can assign them equivalently ordered labels by choosing a suitable order on the set of binary strings. One possibility is to use the usual lexicographical order.[1] Another and sometimes better option is to order strings first by length and then by lexicographical order:

$$\varepsilon, \mathtt{0}, \mathtt{1}, \mathtt{00}, \mathtt{01}, \mathtt{10}, \mathtt{11}, \mathtt{000}, \mathtt{001}, \ldots. \tag{A.1}$$

Lemmas A.3 and A.4 introduce labelings where each label has a length that is proportional to a given weight for each object. In this case, it is useful to have an order on binary strings that evenly intersperses strings of different lengths. The order in (A.1) is terrible in this respect, since all strings of a given length are immediate successors of one another. A better order is the lexicographical order, but this, too, is not ideal since short strings tend to be successors of one another: for example, if a string s is short, so is its successor $s \cdot \mathtt{0}$. We shall here present the order $\preceq$ from [23] which intersperses strings of different lengths quite evenly. The order is, essentially, a lexicographical order in which non-existing bits are considered larger than 0 but smaller than 1. More formally, we define 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 $s, t,'$. Here we have written $s \prec t$ as short for $s \preceq t \wedge s \neq t$. Thus,

$$s \preceq t \iff \begin{cases} \text{both } s \text{ and } t \text{ are empty;} \\ s \text{ is empty and } t = \mathtt{1} \cdot t'; \\ s = \mathtt{0} \cdot s' \text{ and } t \text{ is empty;} \\ s = \mathtt{0} \cdot s' \text{ and } t = \mathtt{1} \cdot t'; \text{ or} \\ s = s_1 \cdot s',\ t = t_1 \cdot t',\ s_1 = t_1 \text{ and } s' \preceq t' \end{cases}$$

The following shows all binary strings of length three or less ordered by $\preceq$:

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

[1] See Section 0.2 in the preliminaries for a definition.

The order $\preceq$ naturally arises in many contexts and has been studied before; see, for example, [40]. Strings of different sizes are quite evenly interspersed when ordered by $\preceq$. In fact, we can construct the order by placing all binary strings on a line, starting with shorter strings and "filling out the holes" with the longer strings in the following way. Start with ε. Then place `0` and `1` before and after ε, respectively, so that we get the order $\texttt{0} \prec \varepsilon \prec \texttt{1}$. Then place `00`, `01`, `10` and `11` before, between and after the three already placed elements, so that we obtain the order $\texttt{00} \prec \texttt{0} \prec \texttt{01} \prec \varepsilon \prec \texttt{10} \prec \texttt{1} \prec \texttt{11}$. And so on. At the i'th step, place the 2^i strings of length i, ordered lexicographically, in the 2^i spaces before, between and after the $2^i - 1$ already placed strings of length strictly less than i.

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

A.1.3 Encodings of proportional sizes

In some cases we are interested in labeling objects according to some specified *weights* such that the label size for each object is inversely proportional to the corresponding weight. If we do not care about the order of the objects, then the simplest solution is just to order the objects descendingly according their weights and assign labels in the same order as in (A.1). However, we can achieve just as good a result, while keeping the order of the objects intact, if we use the order $\preceq$ described in Appendix A.1.2. This is the content of Lemma A.3. The result in Lemma A.4 is a version of Lemma A.3 in which it is ensured that all produced strings are nonempty.

Lemma A.3. *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 $a_i = 0 \cdot b_i$ and for $i > k$ by $a_i = 1 \cdot b_i$. Then (a_i) is a $\prec$-ordered sequence with $|a_i| \leq \lfloor \log w - \log w_i \rfloor$ for all i. □

Lemma A.4. *Given a finite sequence (w_i) of positive numbers with $w = \sum_i w_i$, there exists 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 A.3 to this sequence, one and only 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) of nonempty strings with $|a_i| \leq \lfloor \log(w + w_k) - \log w_i \rfloor$ for all i as desired. □

A linear time implementation of Lemma A.3 can be achieved as follows. First compute the numbers $t_i = \lfloor \log w - \log w_i \rfloor$ in linear time. Now set $a_1 = 0^{t_1}$ 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 Lemma A.3 proves 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 Lemma A.4 can be implemented in linear time.

A.1.4 Encoding lists of strings

Lemmas A.5 to A.8 below show how to encode a list of binary strings into a single binary string. The lemmas assume different restrictions on the total number of strings and describe the length of the encoding in terms of the length t of the concatenation of the substrings. Three of the four lemmas can be implemented with linear time encoding and constant time decoding on a RAM machine in which a machine word has size $\Omega(t)$. Lemma A.6 uses a counting argument together with Lemma A.2 for its proof without actually describing how the counting should be done in practice, and hence this lemma comes without any guarantees for its time complexities.

Lemma A.5. *Let $a = (a_1, a_2)$ be a pair of 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 A.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. □

I thank Mathias Bæk Tejs Knudsen for inspiration to parts of the proof of Lemma A.6 below. As the proof shows, the encoding in Lemma A.6 is optimal with respect to size, but it comes with no guarantees for time complexities. Lemma A.7 further below is a suboptimal version of Lemma A.6 but with a more efficient implementation.

Lemma A.6. *Let $a = (a_0, \ldots, a_{2k})$ be a list of 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 A.2 to encode a for a fixed t. To do this, we must count the number of possible sequences in the described form. 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

$$s_t = \frac{1}{4}c^{t+1} + \frac{1}{4}d^{t+1}, \quad \text{where } c = 2 + \sqrt{2} \text{ and } d = 2 - \sqrt{2}, \tag{A.2}$$

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 A.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 A.2 to recreate a.

It remains to show (A.2). 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 (A.2). □

Lemma A.7. *Let $a = (a_0, \ldots, a_{2k})$ be a list of 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 `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 $a_{2j} \cdot a_{2j+1}$ for some j (which by the assumption is nonempty). The third string has a `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 `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 `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 `1`, then the concatenation should be split up at that position; in particular, if the `1` is at the first bit in the concatenation, then it means that a_{2i} is empty. If none of the bits is a `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 `1`s in the corresponding bits in the third string. If there are two `1`s among these bits, then it is clear how to split up the concatenation. If there are no `1`s, then it means that a_{2k-1} and a_{2k} are both empty. If there is exactly one `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 A.8. *Let $a = (a_0, \ldots, a_k)$ be a list of 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 $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$. 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:

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

By changing base from 3 to 2, we can represent $\tilde{s}$ by a binary string s of length exactly equal to $\lceil (t-1) \log 3 \rceil$. 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. □

A.2 Boxes and groups: a lower bound technique

This section presents a single lemma, which is used throughout the thesis to prove lower bounds on label sizes. The technique, introduced by Alstrup, Bille and Rauhe [5], uses a division of into "boxes and groups" of the set of elements that are to be labeled.

Lemma A.9. *Let X be a set with $|X| = nk$, where n is a power of 2 and $k \leq \log n$. Further, let $e\colon X \to S$ be a function that labels the elements from X with labels from some set S. Assume that have partitioned X into k disjoint subsets of the same size:*

$$X = X_1 \cup \cdots \cup X_k, \quad \text{where } |X_i| = n,$$

and that we have further partitioned X_i into $n/2^i$ partitions of size 2^i:

$$X_i = X_{i,1} \cup \cdots \cup X_{i,n/2^i}, \quad \text{where } |X_{ij}| = 2^i.$$

We call each X_i a box *and each $X_{i,j}$ a* group*. Now, suppose that the following two conditions hold:*

(i) *Two distinct elements of the same box have distinct labels.*

(ii) *If $x_1, x_2, x_1', x_2' \in X$ are elements such that $e(x_1) = e(x_1')$, $e(x_2) = e(x_2')$ and x_1, x_2 belong to two different groups in the same box, then x_1', x_2' belong to two different groups.*

Then $|S| \geq n(k+1)/2$.

Proof. We shall construct a subset X' of size $n(k+1)/2$ in which all elements have distinct labels. The set will be constructed gradually by including or not including entire groups from each box. A group whose elements have been included in x' will be denoted *marked*. Thus, the goal is that all elements of marked groups shall have distinct labels.

We begin by marking all groups in X_k. The property in (i) ensures that all the elements in X_k have distinct labels. We shall mark the groups in the remaining boxes $X_{k-1}, \ldots, X_1$ in this order, and we shall make sure that exactly half of the groups in X_i are marked, for $i = 1, \ldots, k-1$. If a group contains an element

whose label is already in use by an element in a marked group, then we say that the group is *closed*. Else, we say that it is *open*.

Now, assume that we have marked groups from the boxes $X_{i+1}, \ldots, X_k$ and consider the box X_i. Using (ii), we can associate each closed group from X_i with a unique marked group from $X_{i+1} \cup \cdots \cup X_k$: for if two distinct groups in X_i are closed, then the two groups must contain elements x_1 and x_2, respectively, whose labels are already in use by elements x_1' and x_2', respectively, and by (ii), x_1' and x_2' must belong to distinct groups. Thus, the number of closed groups in X_i is less than or equal to the number of marked groups in $X_{i+1} \cup \cdots \cup X_k$. Since we have marked all groups in X_k and exactly half of the groups in $X_{i+1}, \ldots, X_{k-1}$, this number is equal to

$$\frac{n}{2^k} + (\frac{n}{2^k} + \cdots + \frac{n}{2^{i+2}}) = \frac{n}{2^{i+1}},$$

which is exactly half of the groups in X_i. The other half of the groups in X_i are therefore open and can be marked, and the procedure can continue.

When we are done, all groups in X_k and exactly half of the groups in $X_1, \ldots, X_{k-1}$ have been marked. The number of marked elements is therefore $n(k+1)/2$, and we conclude that $|S| \geq n(k+1)/2$. □

Appendix B

A note on unbounded parameters

Section 1.7 briefly discusses the possibility of describing worst-case label sizes as a function of parameters that are unbounded for the graph family in question. For example, one could describe a worst-case label size of $3\lfloor \log n \rfloor$ for Trees even though the parameter n (the number of nodes in an individual graph) is non-constant end even unbounded in the family Trees. The existing literature contains many examples of formulations that are ambiguous in the sense that they present label sizes using the parameter n, even though it is not quite clear if the "n" pertains to a certain subfamily of Trees in which each tree has (at most) n nodes or to a *individual* tree with n nodes. Further, it is often not clear if such a formulation implicitly assumes that the decoder in a labeling scheme "knows" the number of nodes in the graph whose labels it is decoding, and, if so, why this even makes a difference. The purpose of this small note is to clear up some of the confusion.

B.1 Different interpretations

First, recall that this thesis follows the convention of always denoting the number of nodes, the maximum degree and the depth in a *specific* tree by n, δ and d, respectively, and the corresponding upper bounds in a graph family by N, Δ and D, respectively. Thus, for example, a tree with n nodes and maximum degree δ belongs to the family $\mathsf{Trees}(N, \Delta)$ whenever $n \leq N$ and $\delta \leq \Delta$. Second, let us emphasize that we will strictly follow the definitions in Section 1.4 which, in particular, means that a decoder à priori has absolutely no knowledge of the graphs whose labels it is decoding. The decoder knows the labels it receives as input, including their lengths (more on this assumption later), and since the decoder has been constructed with a specific graph family $\mathcal{G}$ in mind, the decoder also knows $\mathcal{G}$ as a whole, including all bounds on parameters there may be for the objects in $\mathcal{G}$. So, for example, if a labeling scheme has been constructed for the family $\mathsf{Trees}(N)$, then the decoder will know N, but it will not know the number n of nodes in a specific tree from $\mathsf{Trees}(N)$ whose labels it is decoding, although it will know that N is an upper bound for n.

So what does it mean if we say that there exists an f-labeling scheme for a graph family $\mathcal{G}$ (for example $\mathcal{G} = \mathsf{Trees}$) with a worst-case label size of at most

$g(n)$ for some function g (for example, $g(n) = 3\lfloor \log n \rfloor$), where "$n$" is used as a parameter even though there are no bounds on n in $\mathcal{G}$? Or, likewise, if we say that any f-labeling scheme for $\mathcal{G}$ has a worst-case label size of at least $h(n)$ for some function h? Consider the following statements:

(1) there exists an f-labeling scheme for $\mathcal{G}$ such that, for any $G \in \mathcal{G}$, the worst-case label size in G is at most $g(n)$, where n is the number of nodes in G;

(2) there exists an f-labeling scheme for $\mathcal{G}$ such that, for any N, its restriction to $\mathcal{G}(N)$ has a worst-case label size of at most $g(N)$;

(3) for any N, there exists an f-labeling scheme for $\mathcal{G}(N)$ with a worst-case label size of at most $g(N)$;

(4) for any f-labeling scheme for $\mathcal{G}$ and any $G \in \mathcal{G}$, the worst-case label size in G is at least $h(n)$, where n is the number of nodes in G;

(5) for any f-labeling scheme for $\mathcal{G}$ and any N, the restriction to $\mathcal{G}(N)$ has a worst-case label size of at least $h(N)$;

(6) for any N, any f-labeling scheme for $\mathcal{G}(N)$ has a worst-case label size of at least $h(N)$;

Statements (1), (2) and (3) concern upper bounds, whereas statements (4), (5) and (6) are their lower bound counterparts. Because of the distinction between n and N as well as between $\mathcal{G}$ and $\mathcal{G}(N)$, we have no problems distinguishing the different meanings of these statements. The literature, however, contains ambiguous statements that could be interpreted as any of (1), (2) or (3).

In general, we have

$$(1) \Longleftarrow (2) \Longrightarrow (3) \quad \text{and} \quad (4) \Longrightarrow (6) \Longrightarrow (5).$$

It is surprising how asymmetric these implications are in the upper and lower bound cases. The fact that (2) implies (1) and (3) follows by restricting an f-labeling scheme for $\mathcal{G}$ to one for $\mathcal{G}(N)$ or by applying it to a specific graph with n nodes. The argument that (6) implies (5) is opposite: if any f-labeling scheme for $\mathcal{G}(N)$ has a worst-case label size of at least $h(N)$, then this must, in particular, hold for any restriction to $\mathcal{G}(N)$ of an f-labeling scheme for $\mathcal{G}$. Statement (4) is almost unrealistically strong and gives a lower bound for label sizes in *any* graph in $\mathcal{G}$, which certainly implies a lower bound on the worst-case label size in the family $\mathcal{G}(N)$, which is the contents of statement (6).

Could any of these implications be equivalences? Certainly, but this requires an argument and depends on the specifics of f, $\mathcal{G}$, g and h. An implication from (1) to (2), for example, could be obtained if $g(n)$ grows with n, so that $g(n) \leq g(N)$ whenever $n \leq N$. To go from (3) to (2) is a bit more involved: the implication could be obtained if we are able to construct an f-labeling scheme

for $\mathcal{G}$ from a collection of f-labeling schemes for $\mathcal{G}(N)$ for $N = 1, 2, \ldots$, which is sometimes, but not always, the case. Likewise, an implication from (5) to (6) could be obtained if we can prove that *any* f-labeling scheme for $\mathcal{G}(N)$ can be constructed as the restriction of an f-labeling scheme for $\mathcal{G}$, which is a strong result and probably not true in many cases. Statement (4) is so strong that it, most likely, is only in very special cases that it can be inferred from (6).

The nonequivalence of (1) with the two other upper bound statements as well as the nonequivalence of (4) with the two other lower bound statements is quite intuitive and due to the difference between considering an individual graph rather than a full family of graphs. An upper bound for label sizes in an individual graph is a "weak" result, since it says nothing about the worst-case label size in the entire family, whereas a lower bound for label sizes in an individual graph is a quite strong result, since it automatically becomes a lower bound for worst-case label size in the entire family.

The nonequivalence between (2) and (3) as well as between (5) and (6) bear witness to an underlying difference between constructing a labeling scheme directly for $\mathcal{G}(N)$ and constructing one for $\mathcal{G}$ and then restricting it to $\mathcal{G}(N)$. Intuitively, the difference is that, when constructing an encoder and a decoder for $\mathcal{G}(N)$ for a specific N, we can exploit the shared knowledge of N in the construction. This is not possible when constructing a labeling scheme that must work for *all* graphs in $\mathcal{G}$, if $\mathcal{G}$ contains arbitrarily large graphs. An encoder surely always "knows" the graph whose nodes it is encoding, but a decoder just receives labels as input without any knowledge of where these labels come from. So a decoder that is part of a labeling scheme for $\mathcal{G}$ does not, à priori, "know" n or any other parameters for the labels it receives, whereas a decoder that is part of a labeling scheme for $\mathcal{G}(N)$ can have the upper bound N built-in. Thus, a decoder made directly for $\mathcal{G}(N)$ is more powerful than one made for $\mathcal{G}$ and restricted to $\mathcal{G}(N)$.

B.2 So what is meant then?

In the preceding, we have argued that there is an essential difference between Statements (1) through (6), and we have mentioned that a substantial part of the existing literature does a poor job in specifying which of these should be the preferred interpretation when stating an ambiguous result. For example, one may find a phrasing such as "*there exists a labeling scheme for trees with a worst-case label size of* $3\lfloor \log n \rfloor$", where it is unclear if the parameter "n" pertains to an individual graph with n nodes (as in (1)) or if it pertains to an upper bound on the number of nodes in the subfamily of trees for which the labeling scheme is constructed (as in (2) or (3)). So which of the statements (1), (2) or (3) should we choose (if any) as the preferred meaning? And why has the literature so widely ignored this ambiguity?

Before we try to answer these questions, let us point out that there are many

variations to the above discussion. So far, we have only considered the number of nodes (n versus N), but one could similarly describe worst-case label sizes in terms of maximum degree (δ versus Δ), depth (d versus D) or other parameters that are unbounded in the graph family $\mathcal{G}$. In fact, one could use more than one parameter in a formula and, for instance, present the worst-case label size for a specific labeling scheme to be $\lfloor \log n \rfloor + \lfloor \log \delta \rfloor$. In this situation, one can again ask if the parameters n and δ pertain to an individual graph or to upper bounds in a subfamily of trees. In fact, there are multiple variations of (2) in this case, since we can consider the restriction to $\mathcal{G}(N, \Delta)$ of a labeling scheme for $\mathcal{G}$, $\mathcal{G}(N)$ or $\mathcal{G}(\Delta)$. So with more than one parameter, there are even more possible interpretations, and hence even more ambiguity.

So let us return to the discussion of which statement is the right one, and why this discussion has been ignored in the literature. The answer to both is, most likely, that it is irrelevant in most cases! Let us begin with the case of upper bounds. Here, it is often the case that knowledge of the bound N simply is not important when designing a decoder for a particular problem, meaning that most labeling schemes in the literature are, in fact, constructed so that they work for $\mathcal{G}$ and not just $\mathcal{G}(N)$. In such a case, (2) holds, and from this follows the two other upper bound statements, whereby it can be left to the reader to choose the formulation of his/her liking. Even if an upper bound formula is clearly stated with n pertaining to an individual graph, the function $g(n)$ is most often increasing with n, in which case (1) and (2) become equivalent. Finally, if it truly is necessary for the decoder to know N, one can always encode N into each label using an additional $2 \log \log N$ self-delimiting bits, and such an increase in label size is often insignificant and likely to "drown" in an $O(\log \log N)$ term.

In the case of lower bounds, (4) is too strong to even be considered a possible interpretation of an ambiguous statement. A proof of a lower bound often involves counting labels that need to be distinct when constructing a labeling scheme for $\mathcal{G}(N)$. Such a counting argument rarely requires that the labeling scheme for $\mathcal{G}(N)$ be a restriction of one for $\mathcal{G}$, and therefore (6) and hence also (5) hold, so the reader is once again free to choose which if these two he/she prefers. So all in all, the literature contains many ambiguous statements, but in practice it rarely makes a difference if we interpret them in one way or the other.

B.3 Knowing label sizes

Even though most decoders presented in the literature do not, à priori, need to know n, they often do need to know the size ℓ of the label they are decoding. One could ask if this assumption is, in fact, based on knowledge of an upper bound N for n, meaning that the labeling scheme must be for $\mathcal{G}(N)$ (as in (3)) rather than for $\mathcal{G}$ (as in (2))? After all, it is not uncommon to assume that whoever reads an encoding of something will know where the relevant code *starts* in the

big hodgepodge of 0s and 1s, but not necessarily where it *ends*, meaning that additional information about the size is needed.

Although this is a valid argument in general, it does not seem reasonable to apply it to labeling schemes, since a requirement that a labeling scheme should work even without knowledge of label sizes is an additional restriction that may very well result in labels that are longer than need be. Thus, if we use extra bits to encode the length of a label, then the bits are wasted if the decoder, for whatever reason, already knows the length. This could happen, for instance, if the labels are transmitted from an external source using a transmission protocol that includes information about the size of the label.

The fact that the designer of a labeling scheme knows nothing about the environment in which the scheme will be implemented and therefore, in particular, cannot know how labels will be made available to the decoder, is a strong argument that we should not add additional restrictions to labeling schemes based on assumptions about things we do not know. Finding out how to store a label in a way that allows it to be correctly retrieved and handed over to the decoder is not a problem for the designer of labeling schemes but for the one implementing them.

B.4 Multiple parameters

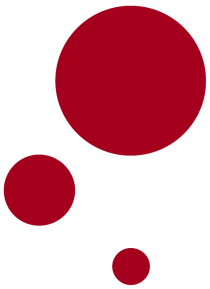
In the above, we have argued that the discussion of whether a parameter is known to the decoder or not is irrelevant in most cases. There are situations, however, where this knowledge is more likely to play a role, namely when multiple parameters are in play. Consider, for example, a labeling scheme for Trees with worst-case label size $\lfloor \log n \rfloor + \lfloor \log \delta \rfloor$. In such a situation it is quite likely that each label is composed of a section of bits pertaining to the variable n and a section of bits pertaining to the variable δ and that the decoder needs to be able to split up the label into these two parts. When the decoder receives a label of size $\lfloor \log n \rfloor + \lfloor \log \delta \rfloor$ it cannot immediately determine how to split up the label into the $\lfloor \log n \rfloor$ first and the $\lfloor \log \delta \rfloor$ last bits. If we replace Trees by $\mathsf{Trees}(N)$, then it may be possible to modify the labeling scheme so that each label has length $\lfloor \log N \rfloor + \lfloor \log \delta \rfloor$, and since the decoder knows N, it knows how to split up a label into its two components. Similarly, if we consider instead the family $\mathsf{Trees}(\Delta)$, then it may be possible to modify the labeling scheme so that each label has length $\lfloor \log n \rfloor + \lfloor \log \Delta \rfloor$, and since the decoder now knows Δ, it knows how to split up a label into its two components. An example of this can be found in Theorem 3.5.

Thus, in cases with multiple variables, it makes a significant difference which of them the decoder knows beforehand, so the difference between the various interpretations is less likely to be irrelevant. Because of the distinction between n, δ and d and N, Δ and D, respectively, there will be no ambiguity when this

thesis presents label sizes in terms of multiple variables. In the literature, however, this is rarely so, and the reader is therefore encouraged to pay extra attention when confronted with label sizes described in terms of multiple variables.

B.5 Conclusion

In conclusion, it would certainly be an improvement to the field of labeling schemes if published results were more specific about what the parameters in formulas actually pertain to. In most cases it is not disastrous to leave the information out, since the different interpretations are equivalent, but in a few cases, especially when multiple parameters are in play, it is essential that it be specified. This thesis does its best to present results without ambiguity.



Bibliography

[1] Serge Abiteboul, Stephen Alstrup, Haim Kaplan, Tova Milo, and Theis Rauhe, *Compact labeling scheme for ancestor queries*, SIAM J. Comput. **35** (2006), no. 6, 1295–1309.

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

[3] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *On finding lowest common ancestors in trees*, Proceedings of the fifth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '73, ACM, 1973, pp. 253–265.

[4] ———, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[5] Stephen Alstrup, Philip Bille, and Theis Rauhe, *Labeling schemes for small distances in trees*, SIAM J. Discrete Math. **19** (2005), no. 2, 448–462.

[6] Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe, *Nearest common ancestors: A survey and a new algorithm for a distributed environment*, Theory of Computing Systems **37** (2004), no. 3, 441–456 (English).

[7] Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen, *Near-optimal labeling schemes for nearest common ancestors*, Submitted to the 25th annual ACM-SIAM Symp. on Discrete Algorithms (SODA), 2014.

[8] Stephen Alstrup and Kasper Green Larsen, *NCA labeling schemes*, Unpublished manuscript.

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

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

[11] Nicolas Bonichon, Cyril Gavoille, and Arnaud Labourel, *Short labels by traversal and jumping*, Electronic Notes in Discrete Mathematics **28** (2007), 153–160.

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

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

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 3rd ed., The MIT Press, 2009.

[15] Reinhard Diestel, *Graph theory, 4th edition*, Graduate texts in mathematics, vol. 173, Springer, 2012.

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

[17] Pierre Fraigniaud and Amos Korman, *Compact ancestry labeling schemes for xml trees*, SODA, 2010, pp. 458–466.

[18] Pierre Fraigniaud and Amos Korman, *An optimal ancestry scheme and small universal posets*, Proceedings of the 42nd ACM symposium on Theory of computing (New York, NY, USA), STOC '10, ACM, 2010, pp. 611–620.

[19] Harold N. Gabow, Jon Louis Bentley, and Robert E. Tarjan, *Scaling and related techniques for geometry problems*, Proceedings of the sixteenth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '84, ACM, 1984, pp. 135–143.

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

[21] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz, *Distance labeling in graphs*, Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms (Philadelphia, PA, USA), SODA '01, Society for Industrial and Applied Mathematics, 2001, pp. 210–219.

[22] R. L. Graham and H. O. Pollak, *On embedding graphs in squashed cubes*, Lecture Notes in Mathematics, Proceedings of a conference held at Western Michigan University, May 10–13, vol. 303, Springer-Verlag, 1972.

[23] Esben Bistrup Halvorsen, *The nearest common ancestor problem*, Master project, 2013.

[24] R. W. Hamming, *Error detecting and error correcting codes*, The Bell System Technical Journal **26** (1950), no. 2, 147–160.

[25] Dov Harel and Robert Endre Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM Journal on Computing **13** (1984), no. 2, 338–355.

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

[27] Sampath Kannan, Moni Naor, and Steven Rudich, *Implicit representation of graphs*, Proceedings of the twentieth annual ACM symposium on Theory of computing (New York, NY, USA), STOC '88, ACM, 1988, pp. 334–343.

[28] Haim Kaplan and Tova Milo, *Short and simple labels for small distances and other functions*, In Workshop on algorithms and data structures, 2001, pp. 32–40.

[29] Haim Kaplan, Tova Milo, and Ronen Shabo, *A comparison of labeling schemes for ancestor queries*, Proceedings of the thirteenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002.

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

[31] 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.

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

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

[34] J. W. Moon, *On minimal n-universal graphs*, Glasgow Mathematical Journal **7** (1965), 32–33.

[35] John Harold Müller, *Local structure in graph classes*, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, USA, 1988, Order No: GAX88-11342.

[36] David Peleg, *Informative labeling schemes for graphs*, In Proc. 25th Symp. on Mathematical Foundations of Computer Science, Springer-Verlag, 2000, pp. 579–588.

[37] ———, *Proximity-preserving labeling schemes*, J. Graph Theory **33** (2000), no. 3, 167–176.

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

[39] Nicola Santoro and Ramez Khatib, *Labelling and implicit routing in networks*, Comput. J. **28** (1985), no. 1, 5–8.

[40] Mikkel Thorup and Uri Zwick, *Compact routing schemes*, Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures (New York, NY, USA), SPAA '01, ACM, 2001, pp. 1–10.

[41] Oren Weimann and David Peleg, *A note on exact distance labeling*, Inf. Process. Lett. **111** (2011), no. 14, 671–673.

[42] Wikipedia, *List of graph theory topics — wikipedia, the free encyclopedia*, 2013, [Online; accessed 15-June-2013].

[43] Peter M. Winkler, *Proof of the squashed cube conjecture*, Combinatorica **3** (1983), no. 1, 135–139.